KOLA:
Knowledge Organization LAnguage

by

Yeona Jang

This research was supported in part by the ITT international Students fellowship program, in part by National Institutes of Health Grant No. R01 LM 04493 from the National Library of Medicine, and in part National Institutes of Health Grant No. R24 RR 01320 from the Division of Research Resources.

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

KOLA:
Knowledge Organization LAnguage

by

Yeona Jang

Abstract

The focus of this research is on a representation of knowledge that captures the structure of a domain into the computational model for efficient retrieval and reasoning. With this desideratum in mind, a concept-based knowledge representation system called KOLA (*K*nowledge *O*rganization *LA*nguage) is described.

KOLA extends the expressive capability of concept-based representation systems by allowing the distinction between definitional and nondefinitional necessary conditions. KOLA allows explicit declarations of properties of relations between concepts (roles or attributes) such as transitivity, symmetry, and so on. The explicit representation of knowledge *about* knowledge helps knowledge to be represented vividly, and reasoning to be performed efficiently. Furthermore, detailed filler references allow instance-specific information to be represented and manipulated effectively.

In KOLA, the terminological reasoning is carried out in a way similar to other concept-based representation systems. The assertional reasoning is performed using an instance network which gets refined, as instances are created or modified. This allows some of assertional reasoning operations to be reduced to the simple graph searching operations.

Keywords: Knowledge Representation System, Frame-based knoweledge representation, concept-based knowledge representation.

Acknowledgment

I would like to express my sincerest thanks to the following:

Professor Ramesh S. Patil, my thesis supervisor, for his kindness, patience, and keen advice throughout this project;

Professor Peter Szolovits, for his insightful advice for this project;

Professor Robert Halstead, for helping me to feel that Tech. Square is a cozy place not a scary as it appeared when I came to U.S.A. and met PPGers;

ITT, for providing the support for me to study in the right place through the international student scholarship program;

Alexander T. Ishii, for his unforgettable, unfeigned help in editing this thesis;

Michael Wellman and Ira Haimowitz, for their helpful discussion about knowledge representation systems;

Dennis Fogg, for his good humor and for helping me to remember that I am a human being who is a social animal;

All members in the Medical Decision Making Group, for providing a pleasant working environment;

My two brothers, for incessantly presenting me the sweet feeling that somebody is caring about me; and

My parents, who believe that I can do whatever I want, without whose support and love I would be lost.

Contents

List of Figures

Chapter 1

Introduction

The objective of this research is to enhance the expressiveness of a concept-based knowledge representation system, while keeping its main desideratum: computational tractability. This means that in KOLA, an expressive limitation still exists as an inevitable result of compromising with computational tractability.

KOLA [1] is derived from KL-ONE which represented knowledge structurally and attempted to distinguish terminological knowledge from assertional. KOLA has the several features that distinguish it from other concept-based knowledge representation systems. In KOLA, an attempt is made to distinguish between definitional necessary conditions of a concept from nondefinitional ones. The distinction between a necessary condition with the transitive property and one without it allows a certain kinds of knowledge to be represented succinctly and manipulated efficiently in KOLA. In addition, using detailed filler references, KOLA's expressiveness and ability to reason with instances is improved.

---

[1]KOLA is the acronym of *K*nowledge *O*rganization *LA*nguage

KOLA consists of three subsystems, C-World, I-World, and Question-Answerer. In KOLA, while reasoning about terminological knowledge is based on the concept taxonomy, reasoning about assertional knowledge is based on an instance network. C-World and I-World are used to build a knowledge base, and have an intelligent user-friendly interface. For example, if they fail to carry out their operations, they show the reason for failures such as the use of undefined concepts, roles, or instances. Such messages are valuable in building a large knowledge base. Question-Answerer is used to obtain information. Question-Answerer performs its operations by retrieving facts or by deducing a limited set of inferences based on them. Information is presented in a stylized way which helps a user to easily get a perspective on what he wants to know.

1.1 Focus

This section describes the limitations of existing concept-based knowledge representation systems which inspired the development of KOLA.

1.1.1 Definitional and nondefinitional information

So far, emphasis in concept-based knowledge representation systems has been placed largely on terminological reasoning. Terminological, categorical knowledge is an important component of a knowledge based system. For example, we need to know whether or not the disease A with some symptoms can also be the disease B whose causes or plausible treatments are known.

In solving real world problems, pure terminological knowledge is not enough.

For example, when teaching a medical student about a disease and its symptoms, a computer tutor may give him/her a description of a patient with this disease. Then, a patient's age may not be included in the given description, because it is not a definitional necessary condition. However, when this disease is suspected in a particular patient, we may need to know how old this patient is to, for example, get a piece of advice about testing or treatment. A patient's age is not a definitional necessary condition, but is likely to be useful in reasoning about an instantiated patient and disease. Currently, concept-based knowledge representation systems has no appropriate method to represent nondefinitional necessary conditions of a concept and, thus, every necessary condition of the concept is represented uniquely without any distinction between them. The usefulness of distinctions between nondefinitional and definitional necessary conditions of a concept and its effect on KOLA's classifier are covered in detail in **Chapter** 5.

If we want to model not only a conceptual part of a domain but also, for instance, a particular person and the particular situation under which he lives, a concept-based knowledge representation system should provide us the facilities to deal with the following: 1) how to tell a system nondefinitional knowledge of a concept and let the system know that such knowledge needs to be manipulated differently from definitional knowledge of the concept; 2) how to tell the system about specific instances of a concept; 3) how to make the system reason about the relationship between an instance and a concept; and 4) how to let the system reason about instances and the relationship between themselves. All of these features are closely related to the assertional power of a concept-based knowledge representation system, and have not been effectively manipulated in the past.

In spite of placing a major emphasis on computational efficiency in reasoning, some of existing concept-based knowledge representation systems adopted the first order predicate calculus to represent assertions and reason about them (see, for example, [Brachman 83] and [Pigman 84]). Therefore, the problems of inefficiency and undecidability in the first order predicate calculus still exists.

1.1.2 Need for a mechanism to reference role fillers

A concept is connected with other concepts by roles. The number restriction of a role is used to specify how many fillers of a role can possibly be filled when a concept is instantiated. Often, some of fillers in the set can be differentiated from others or ordered on the basis of some common property. For example, suppose that the role *Children* of the instance *Jason* are filled with the instances *Kib* and *Brian*. Such references as the oldest-son or first-child can be known explicitly in a domain. The property of such references is that they are instance-specific and, thus, are not generic enough to be defined as a role. Most of existing systems do not have a facility to deal with such information efficiently although it is available in the domain. Telling the system that some of a role's fillers in a particular instance can be reached via a particular reference and making the system use it in a subsequent reasoning process is achieved with the *detailed filler references* in KOLA.

1.1.3 Types of relations between concepts

The possible types of relations [2] between concepts are transitive, symmetric, and inverse.

The existing concept-based knowledge representation systems make no distinction between roles with the transitive property and ones without it. This distinction, however, would facilitate the representation of knowledge, and increases efficiency in reasoning about what is implicit in a knowledge base. The usefulness of such distinction will be described, including

- how it influences the expressiveness,
- how it affects the action of the classifier, and
- how it can improve the reasoning efficiency of a concept-based knowledge representation system.

Symmetry and inverse relation between necessary conditions are also useful as mentioned or implemented in an existing concept-based knowledge representation system. I will focus on describing how they are implemented in KOLA efficiently and how they affect KOLA's capability.

I will also discuss how other features such as synonyms among concepts and a disjointness class of concepts are represented and manipulated in KOLA.

[2] In a concept-based knowledge representation system, relations between concepts are represented as necessary conditions which represent the relationship between concepts.

1.2 Outline

In `Chapter` 2, the criteria that knowledge representation systems should be able to handle and the properties which they should have are described briefly.

The family of concept-based knowledge representation systems based on KL-ONE are briefly surveyed in `Chapter` 3, because the history of knowledge representation systems based on the semantic networks until KL-ONE was discussed well in [Brachman 79], and because KOLA's direct inspiration came from KL-ONE. The primitives of existing concept-based knowledge representations, concepts and roles, are covered in `Chapter` 3.

`Chapter` 4 introduces an example knowledge base which is to be used throughout this paper. Through this example, difficulties in existing concept-based knowledge representation systems are analyzed.

The details of KOLA are described in `Chapter` 5. In `Section` 5.1, C-World which consists of terminological knowledge, including its primitives, distinction between nondefinitional and definitional necessary conditions, distinction between transitive and nontransitive necessary conditions, and the classifier which reflects such distinctions, is described.

In `Section` 5.2, the instantiator which connects an instance with its most specific concepts and builds the instance network is covered. Such connection by an instantiation link is important to perform reasoning about instances: decide which concepts an instance under consideration should belong to. The instance

network in which the structure of an instantiated domain is captured is also accounted for.

`Section` 5.4 describes how Question-Answerer deals with question-answering problems and how other two systems support for Question-Answerer to reach the answer.

Finally, `Chapter` 6 covers conclusions of this project and future works to be done.

Chapter 2

Knowledge Representation Systems

Consider what we usually do, when confronted with a problem. First, we try to make a reasonable decision in order to obtain as correct a solution as possible. Second, as an intelligent agent we justify and use such a decision intelligently. Third, we try to behave thoughtfully and deliberately. It is said that a description of a domain of concern is embedded in our brain, and our behavior is based on our beliefs, desires, morality, and so on. D. Dennett called the system which behaves on the basis of beliefs and desires the intentional system, which he described as follows:

> I WISH TO EXAMINE the concept of a system whose behavior can be – at least sometimes – explained and predicted by relying on ascriptions to the system of beliefs and desires (and hopes, fears, intentions, hunches, ...). I will call such systems ***intentional systems***, and such explanations and predictions intentional explanations and predictions, in virtue of the intentionality of the idioms of beliefs and desires (and hope, fear, intention, hunch, ...). ......

> One predicts behavior in such a case by ascribing to the system *the possession of certain information* and supposing it to be *directed by certain goals*, and then by working out the most reasonable or appropriate action on the basis of these ascriptions and suppositions.
>
> – *Intentional Systems* [Dennet 81]

He defined an intentional system as an agent that can predict and explain other intentional systems' behavior, on the basis of the assumption that an intentional system behaves reasonably, not randomly. AI is aimed at modeling a domain of interest computationally and solving a problem about the domain intelligently, just as a human being does. To make an intelligent, intentional computer system achieve what we anticipate, internalization of knowledge somewhere in the computer system is inevitable. If, however, a knowledge representation system provides for us only the way of describing a domain, it should be less useful than we expect, because there is no guarantee that all of what we are interested in is explicitly represented in a description of the domain. Therefore, a knowledge representation system should itself have an intelligent subsystem, which can perform reasoning services that can draw new conclusions about the world by manipulating knowledge internalized explicitly.

H.J. Levesque accounted for the relationship between KR (Knowledge Representation) and the reasoning system as follows:

> KR is intimately connected with reasoning, because an AI system will almost always need to generate explicitly at least some of what has been implicitly represented.
>
> ......
>
> The basic assumption underlying KR (and much of AI) is that thinking can be usefully understood as mechanical operations over symbolic representation.
>
> – *Knowledge Representation and Reasoning* [Levesque 86b]

2.1 What Knowledge Representation Systems should have

AI researchers have paid incessant attention to representing knowledge in an intelligent agent. In this section, we will briefly describe the requirements of a knowledge representation system. A knowledge representation system should have the following qualities:

- *Descriptive adequacy*: ability to describe knowledge.
- *Compactness*: the smaller the size of a description is, the better.
- *Accessibility*: the ability to retrieve information easily and efficiently.
- *Epistemological Adequacy*: the ability to infer information we want to obtain.
- *Inferential Adequacy*: the ability to perform such inferential operations quickly and efficiently.
- *Acquisitional Adequacy*: the ability to be extended.
- *Primitive set*: the ability to compose whatever we might represent from a small set of basic terms.

Although we want to implement a knowledge representation system with all of these qualities simultaneously, there is the tradeoff between expressiveness and computational tractability. By expressiveness, we mean 1) the ability of a knowledge representation system to express the distinctions necessary for describing domain knowledge, in other words, the ability to describe subtleties, 2)

the ability to avoid undesired distinctions, and 3) the ability to leave some distinctions unspecified in order to allow partial knowledge to be expressed. H.J. Levesque described the necessity of the computational tractability as follows:

> Moreover, for sufficiently expressive representation languages, calculating these implications may be too demanding computationally, and so compromises are necessary. Knowledge representation, then can be thought of as the study of what options are available in the use of a representation scheme to ensure the computational tractability of reasoning.
>
> ......
>
> Secondly, because of the causal role of a KB, it rules out operations that are not computationally manageable. In other words, the operations on a KB need to be semantically coherent without demanding more than what any computer can be expected to do. To better understand these constraints, we need to examine what it means to operate on structures in a way that respects their semantic interpretation.
>
> – *Knowledge Representation and Reasoning* [Levesque 86b]

2.1.1 First order Predicate Calculus

From the early stages of AI, first order predicate calculus has been used to represent knowledge in a computer system. The expressiveness of first order predicate calculus is so powerful that even incomplete knowledge can be represented.

In addition to being used as a knowledge representation system itself, the first order predicate calculus can also be used as an abstract formalization to specify semantics, to analyze and to compare different knowledge representation systems because of its clear semantics [Newell 81].

However, there still is knowledge which may not be captured easily even

in such an expressively powerful representation system: although first order predicate calculus has situational variables, continuous processes such as filling a tea pot with water could not be represented [Hayes85].

Although first order predicate calculus can prove whether an individual with some properties exists in its domain, there is no operator for naming and referring to this individual in order to re-use it in subsequent courses of reasoning. Moreover, some inferential operations on knowledge in first order predicate calculus are undecidable. Much effort has been made to control the efficiency of inferential operations. The reasonable and safe control of reasoning operations in first order predicate calculus is difficult, however, because forcing the control to be redirected or halted endangers its completeness and soundness, and eventually can destroy its clear semantics. A. Newell indicated the limitation of logic as a knowledge representation system in [Newell 81].

Another observation made in first order predicate calculus, which is closely related to the computational efficiency, is that the direct use of first order predicate calculus fails to capture the structure of a domain in representing knowledge. It can be said that first order predicate calculus does not represent the real world appropriately, in the sense that pieces of knowledge represented in the first order predicate calculus are isolated from each other without taking the structure of the domain into account.

2.2 Semantic-Network-based knowledge representation system

It has been recognized that capturing the structure of a domain, the epistemology, is as important as the logical adequacy. With the introduction of the semantic networks, it has become known how the structural organization of knowledge can influence the use of knowledge in reasoning about its domain. The significance of epistemology in a knowledge representation system may be found in [Brachman 79].

Quillian's semantic networks initiated the attempt to organize knowledge in order to allow an intelligent system to manipulate knowledge and to reason about its world more efficiently. The semantic networks should be considered a valuable advance in representing and manipulating knowledge since they allow knowledge to be represented associatively and structurally. M. Minsky pointed out that structural representation of knowledge allows us to understand what is going on fast and to predict what may happen [Minsky 75]. Human-like memory organization provides simple domain inferences cheaply by prepacking knowledge structurally in it.

2.2.1 Its Merits and Demerits

The advantages of organizing knowledge structurally can be examined both psychologically and pragmatically. Such advantages strongly motivate the development of a knowledge representation system in which the structure of a

domain can be embedded.

First, let us focus on the psychological view point. A human being thinks in chunks, and uses knowledge intelligently in order to solve a problem [Sowa 84]. In attacking a problem, we establish plans based on appropriate strategies in a given environment. All of our actions require our intelligence. Our intelligence is reflected in choosing and using knowledge to solve a problem efficiently. While performing each step of a plan, our intelligence should lead us to make proper use of our experiences or the solutions which were already confirmed, instead of repeating previous steps. The structural organization of knowledge with, for example, indexing allows knowledge to be used in this fashion.

Practically, computational efficiency deserves attention, especially in a computer system with limited resources. It may be useless and even dangerous to a patient in the emergency room to require extended period of time to propose a plan for tests or treatment. Maintaining knowledge structurally can make the reasonable control of flow possible, and greatly influence the efficient use of knowledge within a reasonable amount of time. The advocates of semantic networks argue that finding a solution within a reasonable amount of time is as important as the expressiveness of a knowledge representation system.

A number of knowledge representation systems based on semantic networks have been developed and used in AI. Such systems share the following common properties. The knowledge representation system provides the storage[1]

[1] Hardware and the software technologies for memory management allow a large amount of knowledge be stored with reasonable cost and access time.

for knowledge about a domain and efficiently performs a set of inferences over the knowledge encoded. The problem in such semantic-network-based knowledge representation systems is that the cost of computing inferences may be remarkably sensitive to small changes in the expressiveness of the system. Even a modest representation language can prove intractable [2]. Such tradeoff between computational tractability and expressiveness of a knowledge representation system is discussed in [Brachman 85, Moser 83], [Levesque 84], and [Nebel 88].

Frame-based knowledge representation systems are less expressive than first order predicate calculus. Ensuring to solve a problem within a reasonable amount of time requires that only the restricted class of sentences be allowed in a frame-based knowledge representation system.

Another common disadvantage of concept-based knowledge representation systems is insufficient assertional ability. In KL-ONE, assertional capability is limited to asserting statements of existence, establishing statements of coreference of descriptions, and making statements of identity of individual constructs in a particular situation.

KRYPTON, to be described later, provided two types of representation languages, TBox and Abox, in order to exploit advantages of both first order predicate calculus and a frame-based knowledge representation system.

In summary, representation languages based on semantic networks or frames have an intuitive appeal for forming definitional descriptions but are

[2] Complexity of the determination of subsumption relations in a family of frame-based languages is analyzed in [Brachman 84].

severely limited on assertional power, while those based on first order predicate calculus are less limited assertionally but are restricted to primitive, unrelated descriptions.

The two competing factors in knowledge representation systems are *expressiveness* and *tractability*. When we attempt to design, implement, and compare existing knowledge representation systems with each other, the analysis of the computational cost and expressiveness should play an important role. Yet, when we try to build a knowledge representation system, we want it to be expressive and capable of completing its operations in a reasonable amount of time. More details about expressiveness, tractability, and the tradeoff between them may be found in [Woods 86], [Brachman 84], [Levesque 84], and [Nebel 88]
.

Chapter 3

KL-ONE and its Offspring

This section covers the brief history of AI knowledge representation systems which motivated me to implement KOLA. Specially, I will analyze the advantages and disadvantages of KL-ONE and its offspring, KRYPTON and KANDOR.

In 1975, Minsky introduced the concept of *frame* which partitions a semantic network into easily identifiable concepts. His idea, which was adopted from Fillmore's Linguistic case frames as well as from Quillian's semantic networks, was based on the fact that human memory is associated in chunks, and ideas are interlinked. Minsky's frames enable hypothetical situations and relationships between them to be pre-described.

Semantic net-based knowledge representation systems can be divided roughly into two groups: frame-based knowledge representation systems based on Minsky's frame, and concept-based knowledge representation systems which stem from Brachman's KL-ONE. The subsumption relationship defined in KL-ONE-

like knowledge representation systems is different from that in frame based knowledge representations. Throughout this paper, I classify KL-ONE and its offspring as concept-based knowledge representation systems based on such differences. Details of the differences between a concept in KL-ONE and a Minsky's frame can be found in [Woods 86].

FRL [Goldstein 76], Concepts [Lenet 76], KRL [Bobrow 77], UNITS [Stefik 79], and SRL [Fox 79, Wright 84] should be considered as frame-based languages. CAKE which has the layered architecture also employs frames [Rich 85].

In the rest of this section, a brief analysis of concept-based knowledge representation systems is given, in order to provide a perspective on KOLA.

3.1 KL-ONE

KL-ONE, a harbinger of concept-based knowledge representation systems, was developed by *Brachman* [Brachman 79].

3.1.1 Concepts and Roles

KL-ONE consists of a fixed set of epistemologically[1] primitive structure types: *concepts* and *roles.*

A concept defines a class of things in a domain [Moser 83]. A concept does not assert any particular individual in the domain, and thus it does not bear any mention of existence at all. For example, although the Unicorn does

[1]The definition of epistemology is given in [Fox 84].

not exist in our world, we can define and use it as a concept in a subsequent problem-solving process. Each concept consists of necessary conditions for an object to be its member. It can be viewed as a template with a well-defined structure used for modeling a class of things in the domain. Assertions about the domain are made using concepts.

Concepts are divided into two classes – primitive and defined – based on whether or not all necessary conditions for an object to be a member of a concept are specifiable. Considering a concept as primitive means that it is impossible and undesirable to specify all necessary conditions of the concept. Most natural things such as person, apple, tree, *etc*, should be classified as primitive concepts. A concept whose necessary and sufficient conditions can be specified is considered defined.

Another way in which a concept is viewed is based on how many instances for a concept can exist in a domain – individual and generic. A concept which can have only one instance in the domain is defined as an individual concept. A concept for describing, for example, the sun in a solar system is classified as an individual concept. On the other hand, concepts which can have more than one instance in the domain are considered generic.

Roles specifies the logical association between concepts and represent the necessary conditions of a concept. Roles in a concept do not specify default assertions. Therefore, inherited roles in a subconcept cannot be canceled. Roles determine the structure of a concept. In other words, these associations can be used to give a concept its own structure which essentially differentiates it from

its superconcepts. A role can itself be differentiated, forming a role taxonomy similar to a concept taxonomy.

3.1.2 Specialization of concepts and Classifier

Recall that a concept has its own internal structure which differentiates it from its superconcepts. A concept can be defined from more general concepts using the structure-forming operators, inheriting all components of its superconcepts. This means that there is a subsumption relationship between them. The subsumption relationship between concepts is determined computationally by manipulating the essential differences between concepts.

The essential differences between concepts are defined by specializations. A specialized concept can be created either by conjoining two or more concepts, by role restrictions, by role differentiations, or by role constraints. There are two kinds of role restrictions – value restriction and number restriction. Details of concept specialization are discussed in `Section` 5.1.2. Using structural differences between concepts, the classifier can decide mechanically the proximate genus of a newly defined concept and its subsumees. KL-ONE should be considered as an implementation of a structural inheritance taxonomy.

The subsumption relationship is defined as follows. A subsumes B if and only if all instances of B are instances of A. Formally, let A and B be two concepts, and Σ and Ψ be sets of all instances of A and B, respectively. Let b be any element of Ψ. Then,

$$\text{A subsumes B if and only if } \forall b \in \Psi,\ b \in \Sigma$$

We can see that subsumption specifies a necessary set inclusion. The subsumption relationship is transitive and nonreflexive, which imposes a partial ordering on concepts. If C_1 is the subconcept of C_2 and C_2 is the subconcept of C_3, then C_1 is the subconcept of C_3. The classifier uses the transitive property of the subsumption relationship in terminological reasoning.

When a concept is defined, it is placed between its most specific superconcepts and most general subsumees. This process, called classification, is performed automatically by a concept-based knowledge representation system, using a classifier. The result of the classification is the taxonomy of concepts.

The determination of subsumption relations is *NP-hard*, if it is allowed to express quantification or disjunction in the concept-based knowledge representation system, or if the representation of incomplete knowledge is permitted. To ensure that the solution to a problem in a domain is reached in a reasonable amount of time, restrictions are imposed on expressiveness in a current concept-based knowledge representation system, as mentioned in `Section` 2.2.1.

3.1.3 Instances in KL-ONE

An instance is the incarnation of a concept. An instance is a member of a concept, and is assertional in nature. Definitions and assertions in KL-ONE roughly correspond to terms and sentences, or attributive and referential distinction [Martin 81], respectively. While KL-ONE has the power to form descriptions, it has weak assertional ability and, thus, it is not easy to make assertions about a domain. For example, there is no appropriate way to tell the system about

instances in KL-ONE. An instance is treated as an individual concept in KL-ONE, which blurs the distinction between concepts and instances. Representing an instance as a concept is inconsistent because a concept is only a definition of a term. Treating an instance as a concept makes an instance to be included inappropriately in the terminological box. However, we want to keep the terminological box as generic and succinct as possible.

3.1.4 Reasoning in KL-ONE

In order for a knowledge representation system to be useful, it must have the ability to reason about its domain as well as represent knowledge. KL-ONE, as a knowledge representation system, includes an inference mechanism for drawing the consequences of the use of descriptions. Inferential capabilities of KL-ONE are provided by the classifier. The primary function of the classifier is to find relationships among concepts and to construct a concept taxonomy by comparing their structures. The classifier, then, carries out terminological reasoning about concepts based on the concept taxonomy built: for example, reasoning about concept subsumption.

On the other hand, as discussed in `Section` 2.2.1, KL-ONE has limited assertional ability: its assertional capability is limited to asserting statements of existence, establishing statements of description coreference, and making statements about the identity of individual constructs in a particular situation.

3.1.5 Summary

Although the assertional power of KL-ONE is weak, it attempts to distinguish definitional terminological knowledge from assertional knowledge. Such a distinction of terminological knowledge from assertional knowledge can improve the aesthetics of knowledge representation. First order predicate calculus does not make this distinction, and even definitions are expressed as predicates, just like assertions. KRYPTON and KANDOR, both of which are descended from KL-ONE, have the definitional component for analytical knowledge and the assertional component for synthetic knowledge.

The details of topics discussed in this section may be found in [Brachman 85], [Patel-Schneider 84], [Moser 83], and [Pigman 84]

3.2 KANDOR

KANDOR is a small concept-based knowledge representation system designed to provide important services to the rest of the knowledge-based system[2]. Based on the idea that small can be beautiful [Patel-Schneider 84], the expressiveness of KANDOR is limited to ensure that inferences, which are specially helpful in constructing appropriate queries and in efficiently retrieving individuals that match a query, are performed in a reasonable amount of time. KANDOR has limited expressive power, but its inferential procedures are sound and complete

[2]KANDOR was used as the knowledge representation part of ARGON [Patel-Schneider 84] that is the system for retrieving information efficiently and effectively in reasonable time and for guiding a user to get the information he/she needs without getting lost his/her way in this heterogeneous knowledge-based system.

as well as fast.

The basic units of KANDOR are *individuals* and *frames*. A frame (a concept in KL-ONE) is, in essence, a specification of conditions that an individual must meet if it is to be considered as the frame's instance. A frame is defined by giving a list of more general frames and a list of restrictions. In KANDOR, both value restrictions and number restrictions are provided in order to define a new frame, and a classifier is used to construct the frame taxonomy. The subsumption relationship between frames in KANDOR is similar to the subsumption relationship between concepts in KL-ONE. The frame taxonomy of KANDOR is strict, and thus it is possible to give a semantic account of frame subsumption. The details of KANDOR can be found in [Patel-Schneider 84, Pigman 84].

An individual (an instance in KL-ONE) is used to assert objects in the real world. Individuals are associated with other information by means of *slots*, which map every individual into slot fillers. In addition to the classifier, KANDOR has a *realizer* which, when an individual is created, connects it with the most specific frame that it is an instantiation of.

There are two main differences between KL-ONE and KANDOR in representing and manipulating knowledge. One difference is found in the value restriction. In defining a new frame, the value restriction of a slot in KANDOR not only can be a frame but also an individual which asserts an existing entity in a domain. In KL-ONE, the value restriction of a role is only a concept in which there is no mention of existence at all. It is not consistent to use an assertional value in a conceptual description, although using it in defining a class of objects

seems technically easy and plausible. Another difference is that while there is no proper way to define instances in KL-ONE, KANDOR can define, and even manipulate them using its realizer.

In KANDOR, however, there is some difficulty arisen in dealing with number restriction and an individual's slot fillers. Suppose we have the frame *Person* with the slot *Name* which has a minimum number restriction to some value, say **x**. To be an individual of the frame *Person*, the individual has to have at least **x** fillers in the slot *Name*. Although knowledge about an individual bears enough information to say that it is a member of the frame *Person*, the realizer cannot deduce that the individual is a member of the frame *Person*, if less than **x** fillers of the slot *Name* are specified and we do not tell the system that it is a member of the frame *Person* explicitly. It, however, may be possible that we are sure that this individual has some name which is unknown at the time when a frame is instantiated. The realizer should recognize that this individual is a member of the frame *Person* and to use it as an instantiation of the frame *Person* in a subsequent reasoning process.

In summary, although KANDOR is small, it is fast. KANDOR has both a *classifier* to manipulate the subsumption relationships between frames and a *realizer* to deal with the relationship between individuals and frames.

3.3 KRYPTON

KRYPTON was implemented to combine the advantages of predicate calculus and concept-based languages.

KRYPTON is composed of two types of knowledge representation languages: a concept-based one for forming the descriptive and structured definitions for terminological information (*TBox*), and a logic-based one for making assertions of contingent facts about a world of interest – facts in a given domain that happen to be true, but are not necessarily true (*ABox*) [Pigman 84]. In KRYPTON, a taxonomy containing frame-like definitions is integrated with a non-causal connection graph theorem prover [Brachman 83, Stickel 83, Stickel 82].

TBox language is used to represent terms and captures the essence of frames within a *compositional* and *strictly definitional* framework, without the ambiguities and possible misinterpretations common in existing frame languages.

There are two types of expressions in TBox: *concept* and *role* expressions. Their definitions are similar to those for concepts and roles in KL-ONE, except that KRYPTON has no structural descriptions and number restrictions.

In TBox, a terminological hierarchy is constructed on the basis of a partial ordering on the subsumption relations. The subsumption and disjointness relationships among terms in TBox are determined by their structures, not by any domain-dependent facts. So, we can ask TBox analytic questions about the hierarchy such as:

Is X an instance of C? or

Is C1 subsumed by C2?

ABox language is used for representing sentences and enables a user to state facts about a domain. Using ABox, a user can build descriptive theories. ABox uses Stickel's non-clausal connection graph resolution theorem prover by employing information from TBox [Stickel 82]. ABox is able to reason with incomplete information, deal with quantification, and make use of terminological information in TBox. While TBox knows only definitions, ABox knows everything else.

KRYPTON, however, is very large and has all the demerits of first-order predicate calculus because the theorem prover component which uses resolution is based on logic. Even though it improves on other theorem provers by incorporating TBox in its reasoning, the theorem prover is, nevertheless, limited typically to dealing with domains which are both highly structured and highly constrained, and requires an enormous amount of control for inferential operation to solve a problem [Nilsson 80].

In summary, KRYPTON is a functionally hybrid knowledge representation system, so that assertional reasoning as well as terminological reasoning can be performed. In TBox which consists of concepts and roles, terminological knowledge about its domain is embedded and reasoned with. In ABox which consists of assertions, first order predicate calculus and the nonclausal connection-graph resolution theorem prover are adopted to manipulate these assertions. Thus, KRYPTON is also subject to the disadvantages of first order predicate calculus,

while it has better assertional ability than KL-ONE or KANDOR. The details of KRYPTON can be found in [Pigman 84].

Chapter 4

Example and Observations

This chapter includes the running example used for explaining what I attempt to achieve through this research. The example shows us the difficulties which arise in representing knowledge in the existing concept-based knowledge representation systems: it will be discussed the details of how KOLA deals with these difficulties in `Chapter` 5.

4.1 Example for explanation

Suppose we are going to build a knowledge base in a concept-based language. The complete knowledge base which describes the example domain is found in `Appendix` A.4. For explanation's sake, a part of this knowledge base, shown below, is to be analyzed.

```
1) Kidney is a part of Urinary system
2) Nephron is a part of Kidney.
```

```
3) Nephron-Disease is the specialization of Kidney-Disease.
4) Jason's wife is Mary.
5) Jason has two Children:  Kib and Brian.
6) Jason is the surgeon.
7) Kib is the first son of Mary.
8) Brian is the second son of Mary.
9) Brian has the nephrotic disease.
```

The first three facts are terminological, while the rest of given facts are assertions.

Figure 4.1 is the graphical representation of the example in a concept-based knowledge representation system. The conventional pictorial notations are used: an oval for a concept, a double arrow for subsumption relation, and a single arrow with the encircled square for a role. Only terminological knowledge is shown in the figure. Assertional knowledge is not shown in this figure, because no appropriate graphical notation for representing knowledge about instances is provided in existing concept-based knowledge representation systems. This motivates me to develop new pictorial notations for assertional knowledge in KOLA.

A user can ask questions such as *"Is a kidney anatomical part of urinary system?"* or *"Who are (is) Jason's children?"* of the knowledge base. This type of questions can be answered efficiently by searching through the structure embedded in the knowledge base. The details of how it is performed may be found in [Brachman 85], [Patel-Schneider 84], [Moser 83], and [Pigman 84]. Then, what are the difficulties?

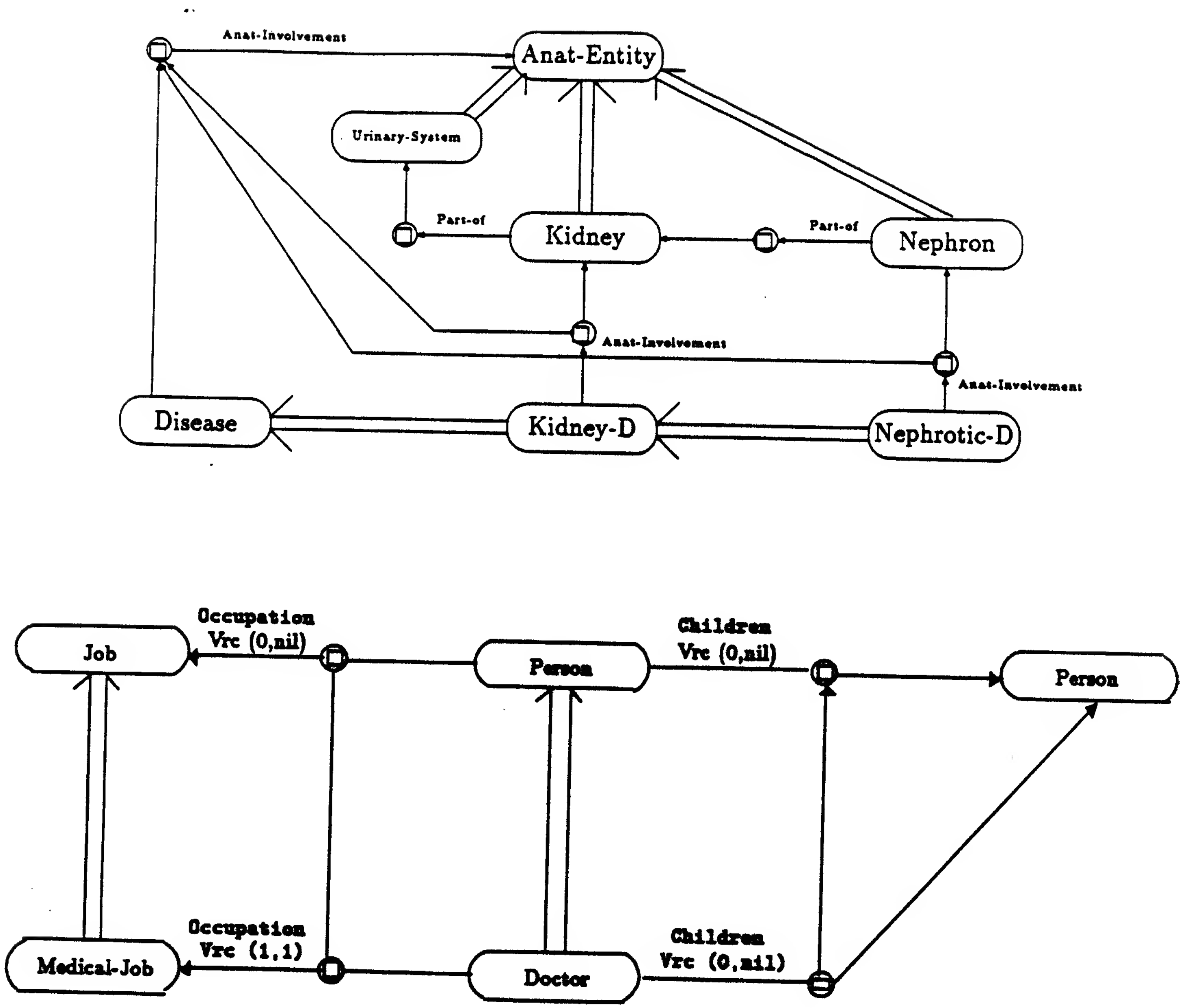


Figure 4.1: Pictorial Description of Example

4.2 Difficulties encountered

Consider again the nine example facts given in Section 4.1. While facts 4), 5), and 6) can be represented and manipulated efficiently in previous concept-based knowledge representation systems, several difficulties arise in representing the rest.

4.2.1 In Terminological Knowledge

The problem in representing the facts 1), 2), and 3) (for simplicity, referred to as *FACTS* in this section) is how to deal with the *Part-of* relation efficiently. The following examples, presented at *NIKL Workshop* (1986) , demonstrate the difficulties involved in this problem. Method [1] is suggested by Patil, and method [2] by Schmolze.

```
[1]  (defrole PART-OF primitive
          (domain ANAT-PART)  (range ANAT-PART))
    (defrole ANAT-INVOLVEMENT primitive
          (domain DISEASE) (range ANAT-PART))

    (defconcept KIDNEY primitive (specializes ANAT-PART))
    (defconcept NEPHRON primitive (specializes ANAT-PART)
        (res PART-OF (vrc KIDNEY)))
    (defconcept KIDNEY-DISEASE primitive (specializes DISEASE)
        (res ANAT-INVOLVEMENT (vrc KIDNEY)))

    (defconcept NEPHROTIC-DISEASE primitive
```

```
          (specializes KIDNEY-DISEASE)
          (res ANAT-INVOLVEMENT (vrc NEPHRON)))

[2]  (defrole PART-OF primitive
          (domain ANAT-PART)  (range ANAT-PART))
     (defrole ANAT-INVOLVEMENT primitive
          (domain DISEASE) (range ANAT-PART))

     (defconcept KIDNEY-PART primitive (specializes ANAT-PART))
     (defconcept KIDNEY primitive (specializes KIDNEY-PART))
     (defconcept NEPHRON-PART primitive (specializes KIDNEY-PART))
     (defconcept NEPHRON primitive (specializes NEPHRON-PART))
     (defconcept KIDNEY-DISEASE primitive (specializes DISEASE)
          (res ANT-INVOLVEMENT (vrc KIDNEY-PART) (min 1)))
     (defconcept NEPHROTIC-DISEASE primitive
          (specializes KIDNEY-DISEASE)
          (res ANAT-INVOLVEMENT (vrc NEPHROTIC-PART) (min 1)))
```

– *1986 NIKL Workshop* [Personal Communication]

We want to define that a disease of an object is also a disease of any other object of which this is a part. In method [1], however, after classifying the role *ANAT-INVOLVEMENT* and the concept *NEPHROTIC-DISEASE*, the classifier constructs the new concept, made by conjoining *KIDNEY* and *NEPHRON*, and assigns it as the range of *ANAT-INVOLVEMENT* in the concept *NEPHROTIC-DISEASE*. This is not what we desire. Even though a person has a disease in his nephron, his disease cannot be an instance of the concept *NEPHROTIC-DISEASE*, since if a disease is to be a member of *NEPHROTIC-*

DISEASE, the filler of *ANAT-INVOLVEMENT* has to be an instance of the conjoined concept *KIDNEY* $\wedge$ *NEPHRON*.

Consider method [2]. Patil's response was that modeling the domain in this fashion was undesirable, because the representation begins to resemble a programming language. In general, it was argued that a knowledge engineer has an idea of how he/she represents the domain, and a knowledge representation system should provide him/her the way to represent it naturally. In addition, because the concept *KIDNEY-PART* represents the set of all parts which consist of the kidney, relating the concept *KIDNEY* to *KIDNEY-PART* with some role which specifically represents this fact would be more accurate than relating them with subsumption.

In modeling the domain with *FACTS*, method [1] is preferable to method [2] if some mechanism is provided to overcome the difficulties we have described. The mechanisms in KOLA to deal with these difficulties are described in **Chapter** 5.

Consider the following questions which requires the manipulation of *FACTS*:

1. *"Is the nephron a part of the urinary system?"*

2. *"Is kidney the anatomical involvement of Brian's nephrotic disease?"*

When manipulating *FACTS*, we can use the extra implicit knowledge that *Part-of* is transitive. Thus, we draw logically the conclusion that the nephron is a part of the urinary system now that the nephron is a part of the kidney and the

kidney is a part of the urinary system, and thus the nephron is a part of the urinary system. [1] We can also respond that Brian has a disease in the kidney that the nephron is a part of, because he has a nephrotic disease. We know that *part-of* is transitive, and use it in answering the given questions.

The existing concept-based systems cannot answer queries 1 and 2 correctly using the knowledge represented in [1] or [2], because they do not have the facility to draw the new conclusion (for example, the kidney is the anatomical involvement of Brian's disease).

The problem observed in answering these queries is applicable to any role which has transitive property. KOLA has the facility to be told explicitly the fact that a role is transitive, and manipulate it efficiently in a subsequent reasoning process.

4.2.2 In Assertional Knowledge

The system should know about instances which exist in a domain to reason about the domain. In KL-ONE, an instance is treated as a kind of concept, as discussed in `Section` 3.1.3. On the other hand, KANDOR has a realizer for defining and manipulating an instance, albeit it has the problem in handling an instance which has a role with the minimum number restriction (see `Section` 3.2). KOLA has an instantiator, similar to a realizer in KANDOR, which can

[1] In question 1, we assume that it is asking for the anatomy of a human being not that of a particular person. Surely, a question such as *"Is Jason's kidney a part of Mary's urinary system?"* can be asked: the answer for it depends on whether or not Jason's kidney has been transplanted to Mary's urinary system. Although a knowledge representation system needs to deal with this kind of problem, it is another issue and our assumption is sufficient to account for the observation which is to be given: in deed, KOLA can handle this type of questions.

deal with this problem (details in `Chapter` 5).

In dealing with an instance, we also need a mechanism to efficiently represent references which are not generic enough to be defined as a role of a concept, but are useful in reasoning about an instance, as mentioned in `Section` 1.1.2. For detailed explanation, consider knowledge such as **`Kib is the first son of Mary`** or **`Brian is the second son of Jason`**. Such knowledge should be useful to reason about Mary's children. Without representing such knowledge efficiently, we may get information about her children at the expense of the following inefficiencies.

In the example, Jason's *Children* role has two fillers: Kib and Brian. In the existing knowledge representation system, we can represent only that Jason has two children, and the only information we can obtain is his two children from *Jason*'s *Children* role. Even though information about Jason's first son is available, it is not easy to tell efficiently to the system about it in previous concept-based knowledge representation systems.

Knowledge such as **`Jason's first son is Kib`** can be represented in the following way: instead of defining a role corresponding to children in the concept *Person* itself, we postpone defining it until its instance is created. In other words, the concept *Person* does not have the role *Children*, and its instantiated person has roles for representing knowledge about children. It is conceivable to do so in KL-ONE, because an instance is treated as a concept. We, however, may sometimes need to manipulate the generic information about a person's children regardless to the number, the order, or the gender of children. There

may be the operations which are commonly applicable to children of any instance for the concept *Person*. Then, it is reasonable and more efficient to define the role *Children* in the concept *Person* and to attach such operations to it.

Another method plausible is to differentiate the role *Children* in the concept *Person* into *first-son, first-daughter*, $\cdots$. But how can we possibly decide how many sons and how many daughters the concept *Person* has without mentioning a particular existing person? The concept *Person* will be used as a concept not for a single instance but for a number of instances. The number of male-, and female- children cannot be decided until the concept *Person* is instantiated because it depends largely on a particular instance.

A third method is that the concept *Person* has the role *Children* and when its instance is created, the inherited role *Children* can be differentiated to appropriate roles to embed knowledge about the instance's children. Unless an instance is treated as a concept like in KL-ONE, this method is not applicable.

Suppose that an instance could be treated as a concept. A role in a concept-based knowledge representation system has its own internal structure which is formed by means of differentiation. We can represent it as follows: for the concept *Person* which has the role *Children*, the role *Children* is differentiated into subroles such as *First-son* or *Third-daughter* in a subconcept of the concept *Person*, not in the concept *Person* itself. However, the problem is that the differentiation of a role in this way causes the undesirable computational problem in determining the subsumption relationship between roles, i. e. building the role taxonomy. Whenever an instance for the concept *Person* is created

and its role *Children* is differentiated, the role *Children* has different internal structure. The disjunction of a role's structures should not be allowed because it causes the *NP-hard* problem in the determination of subsumption relationship among concepts. The reason why the disjunction of roles is not allowed in terms of computational tractability can be found in in [Brachman 84] and [Levesque 84].

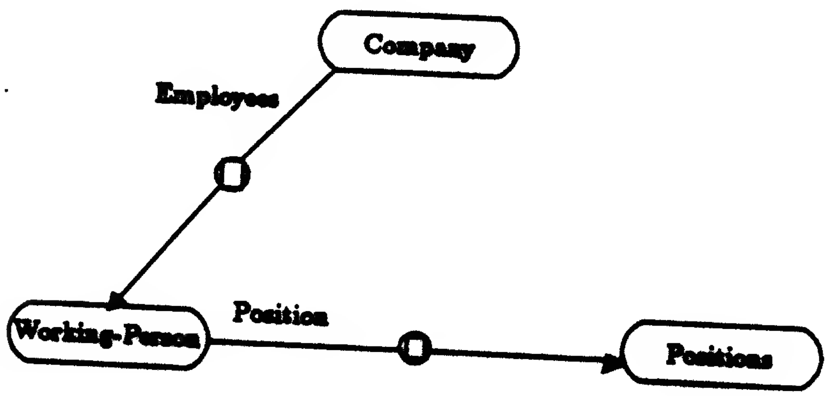


Figure 4.2: Relationship between Company and Working-Person

Defining a particular role such as *Position* to represent *First-son* or *Second-son* is another method conceivable. However, this is inefficient as shown in the following. Suppose that we have the concept *Company* which has the role *Employees* whose value restriction is the concept *Working-Person*. The concept *Working-Person* has the role *Position* in order to represent the information about president, vice president, *etc.* Figure 4.2 shows the relationships between concepts of concern. Suppose that for an instance of the concept *Company* which consists of hundreds of workers, our question is *Who is the first*

president of the company?. For solution, the problem solver has to go through (in the worst case) all workers in the company, checking to see if the value of the role-filler of the role *Position* in every employee is *first-president*. If knowledge were represented directly, such operations would not have to be performed. Such knowledge needs to be represented more succinctly.

In summary, from the system's perspective, the methods prolong unnecessarily determination of the place of a role in the role taxonomy or require extra inferential operations. From a knowledge engineer's perspective, he is forced to write a long segment of statements to represent even simple knowledge. We need a knowledge representation system which has the ability to both express knowledge succinctly and solve the problem efficiently: KOLA achieves it with detailed filler references.

Chapter 5

KOLA

In the first four chapters, the difficulties encountered in the current concept-based knowledge representation systems were discussed: 1) the distinction between definitional necessary conditions and nondefinitional necessary conditions, 2) the transitive property of a (at least) necessary condition, and 3) instantiation of a concept and detailed filler references. In this chapter, it is discussed how KOLA handles such difficulties. The contribution of overcoming such difficulties to AI knowledge representation systems is also described.

KOLA consists of three subsystems: C-World, I-World, and Question-Answerer. C-World [1], which roughly corresponds to TBox in any other concept-based knowledge representation system, contains not only terminological knowledge but also knowledge about terminological knowledge itself. C-World memorizes knowledge about terminological knowledge, so that Question-Answerer

[1]C-World is coined from the fact that concepts and the classifier are the main props of this subsystem.

may use them for reasoning about its domain. I-World [2] contains the assertions made by using terminological knowledge in C-World about the domain, and knowledge about such assertions. Question-Answerer has the ability to answer questions by manipulating knowledge in C-World and I-World appropriately.

5.1 C-World

H.J. Levesque pointed out why terminological knowledge needs to be manipulated separately from assertional knowledge as follows:

> In order to behave knowledgeably in a real domain, a system will have to interact with experts using specialized terms .... Therefore, the application of knowledge representation to expert problems demands of a representation system the ability to *develop, augment,* and *maintain* this kind of technical vocabulary.
>
> *H. J. Levesque* in *Competence in Knowledge Representation*

5.1.1 Primitives

KOLA uses extended versions of primitives and components of KL-ONE – including concepts, roles, subsumption relationships, inheritance, as covered `Section` 3.1. In this section, I focus on how KOLA's features are different from KL-ONE's.

In a concept-based knowledge representation system, one major difficulty lies in representing terminological knowledge. Living in a flood of information, we may need to know thousands or millions of concepts, not just a single one.

[2] I-World is coined from the fact that instances and the instantiator are the main props of this subsystem.

To use such an enormous amount of information, essential descriptions about objects should be stored as succinctly as possible. We should avoid including incidental information in defining a concept.

Yet, there is a need to include incidental knowledge in a knowledge base. To do this, a system should be able to distinguish definitional knowledge on a concept from nondefinitional one, and manipulate such distinction appropriately. Necessary conditions of a concept have been abused, when used to represent nondefinitional as well as definitional necessary properties. The distinction between definitional necessary conditions and nondefinitional necessary ones has not received enough attention in the literature. I would like to discuss the problems that arise due to the lack of distinction between them.

Ronald Brachman defined a role in KLONE as follows:

> The roles represent the various kinds of attributes, parts, etc, that things in the world are considered to "have". These include, for example, such things as parts (e. g. , fingers of a hand), inherent attributes of objects and substances (e. g. color), arguments of functions (e. g. multiplier and multiplicand of a multiplication), and "cases" of verbs in sentences (e. g. "agent"). Any generalized attribute of this sort has two important pieces: (1) the particular entity that becomes the value for the attribute in an instance of the Concept, and (2) the functional role which that entity fills in the conceptual complex. A Role is a formal entity that captures both of these aspects in a structured way, by packaging up information about both the role filler and the functional role itself.
>
> – *On the Epistemological Status of Semantic Networks* [Brachman 79]

Roles were used to define whatever properties things in the real world can have, and there was no distinction between definitional and nondefinitional

conditions in KLONE.

No appropriate method to represent and manipulate a nondefinitional necessary condition of a concept has been provided with any existing concept-based knowledge representation system. In such a system, nondefinitional necessary conditions of a concept as well as definitional necessary ones are included in the concept's structure. They are manipulated identically to represent knowledge about concepts and to reason about the domain. Such a definition of a concept may prevent the concept from being classified correctly, and slow down the terminological reasoning. We may also receive an unnecessarily long description when we ask the system information about a concept.

When we create an instance for a concept, we may get information on nondefinitional necessary conditions. Instead of ignoring such information, we should want to let the system know about it, even though it is not definitional: now that it can be used efficiently for solving a problem about assertional things. For example, a patient's age may not be the definitional necessary condition to define a patient. But in the reasoning session, a patient's age may be critical to diagnose his disease. We need the efficient way to tell the system about such information.

On the other hand, if we impose a restriction so that a concept may consist of only definitional necessary conditions, it is impossible to represent nondefinitional information about an instance unless it is defined in the instance level as in KL-ONE. In KOLA, the distinction between a concept and an instance is strict. An instance is strictly an instantiation of a concept (the definition of

instantiation is found in Section 5.2.1). An instance in KOLA is not treated as an individual concept. Thus, a role cannot be defined when an instance is created. This implies that a concept needs to have roles for representing nondefinitional information as well as definitional one.

To simultaneously obtain essential information about a concept and tell a system nondefinitional information, the distinction between definitional roles and nondefinitional roles is necessary. In KOLA, necessary conditions of a concept are divided into two types – definitional and nondefinitional. Definitional conditions are called roles, while nondefinitional conditions are called attributes. Though both definitional and nondefinitional necessary conditions are included in a concept's structure, they are manipulated differently in KOLA. For example, during a request for the description of a concept, KOLA will suppress nondefinitional information, unless a user asks for it explicitly.

It is hard to decide whether or not a necessary condition is definitional. KR researchers as well as philosophers have argued it. When we are building a knowledge base, one heuristic to decide its definitionality is to view a necessary condition which, by the consensus of experts with thorough understanding of a domain, needs to be included in a description of a term.

Pictorial Conventions

For easier comprehension, knowledge in a concept-based knowledge representation system can be represented in pictorial form, as mentioned in Chapter 4. KOLA uses conventional pictorial notations that are rich enough to deal with the distinction of roles and attributes: a role, or definitional (at least)

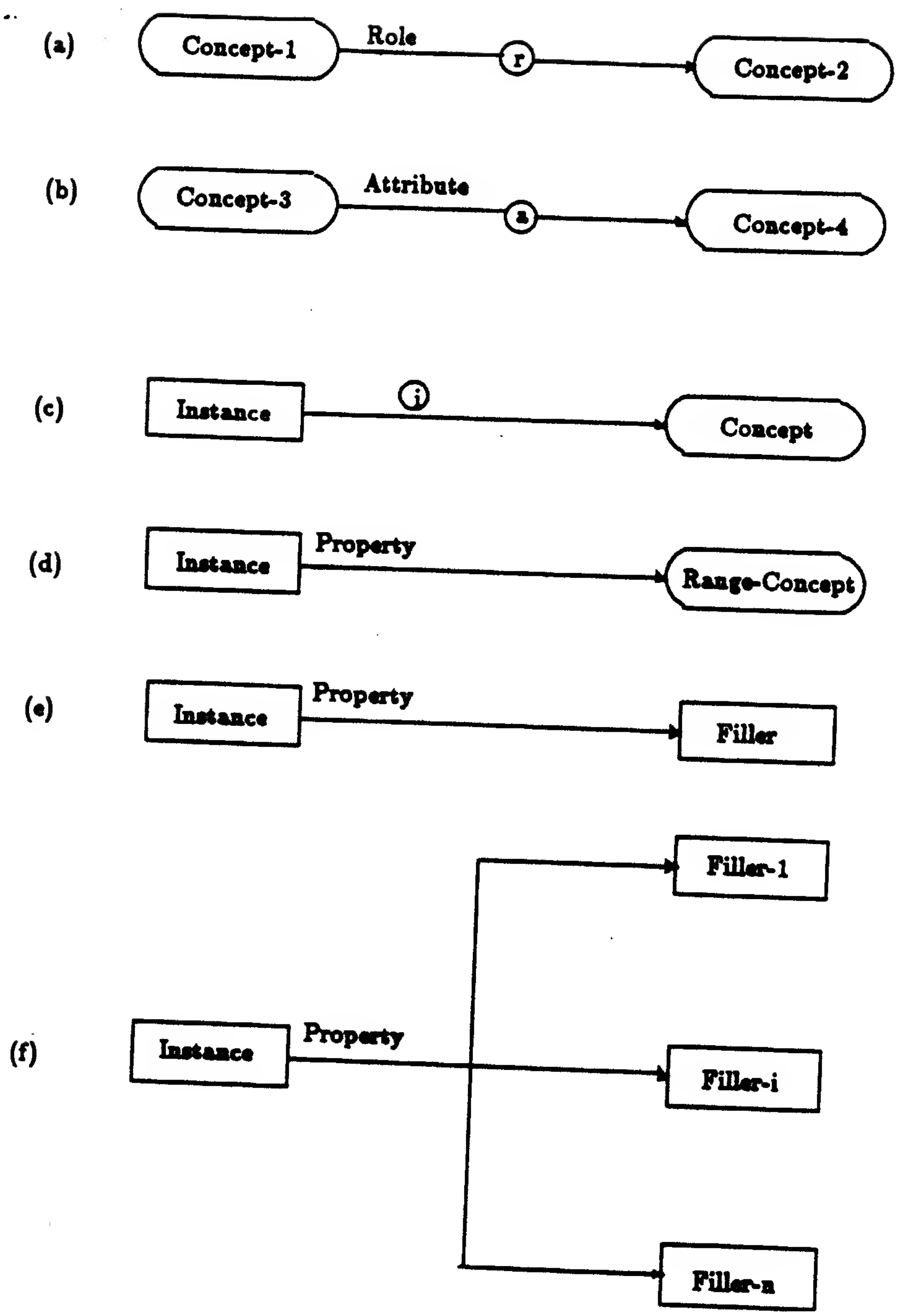


Figure 5.1: Graphical Notations of primitives and its relations in KOLA

necessary condition, is represented by a single arrow with encircled r, while an attribute, nondefinitional condition is represented by a single arrow with encircled a. Figures 5.1 (a) and 5.1 (b) show concepts and the relationship among them via roles or attributes. A link between two concepts is labeled with the name of a role or attribute.

5.1.2 Subsumption Relationship and the Classifier

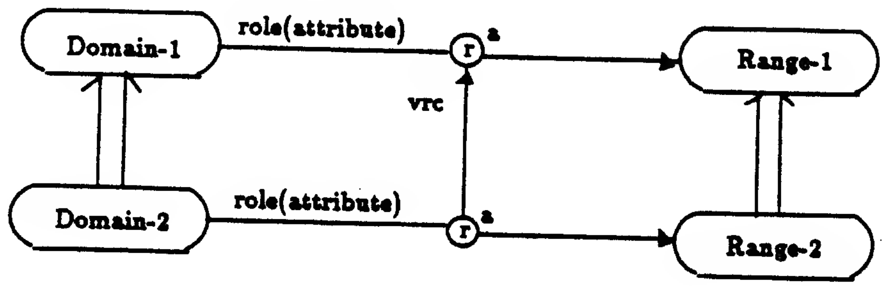
As already mentioned in Section 3.1.2, a concept can be defined by more general concepts. In KOLA, when a concept is defined from more general concepts, it inherits every necessary condition regardless of its definitionality, just as a concept in KL-ONE inherits every role in its superconcepts. Thus, an inherited necessary condition of a concept cannot be canceled.

That a concept can be defined by more general concepts means that there exists the subsumption relationship among concepts. The definition of the subsumption relation was covered in Section 3.1.2. This definition shows that the subsumption relation is determined by extensions of concepts, i. e., the subset relationships between sets of instances. We need, however, to compute the subsumption relationship without waiting for all instances possible to be created, because there can be in a domain a concept with an infinite set of instances or a concept without any instance in a domain.

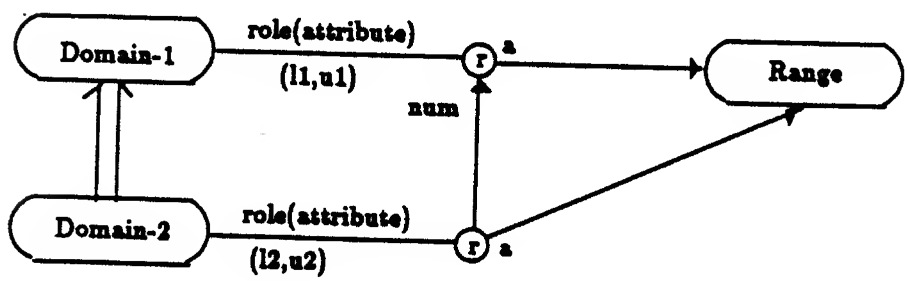
We can compute the subsumption relation by manipulating the structural differences between concepts. In KOLA, a concept is defined in a way similar to that in KL-ONE, but more refined.

Figure 5.2 shows some of the subsumption relationships in KOLA: single arrows labeled with vrc, num, and AtoR indicate value, number, and attribute

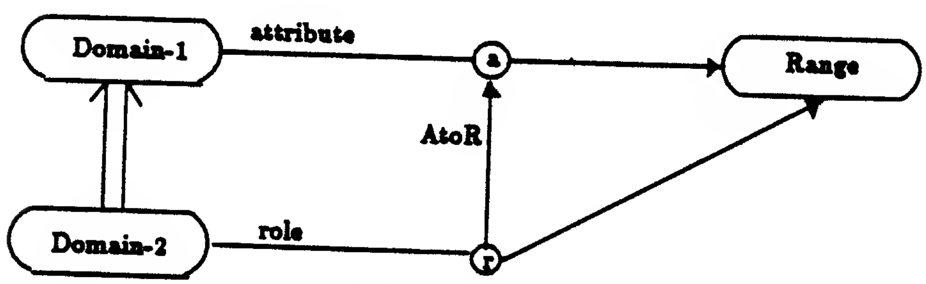
to role restrictions, respectively. We can define a new concept with appropriate combinations of the methods explained below.



(a) Value Restriction



(b) Number Restriction



(c) Attribute to Role Restriction

While cases (a) and (b) are the same as those in KL-ONE, case (c) creates a new concept by converting a nondefinitional necessary property to a definitional.

Figure 5.2: Subsumption relations possible between concepts

1. Declaration as a primitive concept:

   We define primitive concepts by declaration. Such primitive concepts are used as the foundation of a knowledge base.

2. Conjunction of concepts:

   A concept can be defined as the conjunction of other concepts. Such a concept becomes a subconcept of each of the concepts in the conjunction. The conjoined concept inherits every role and attribute from each of its superconcepts, and none of the roles and attributes inherited are canceled.

3. Defining new necessary conditions:

   A concept can be defined with new necessary conditions which any of its superconcepts does not have, as well as with inherited necessary ones. It needs to be strongly emphasized that when a concept is defined by this method, it implies that none of its superconcepts has the newly defined necessary condition as its necessary condition. Therefore, when we define a new necessary condition, we should be careful to place a necessary condition in the most general concepts which can have it as a role or an attribute.

4. Value Restriction:

   When a concept is defined with more general concepts, the range of an inherited role (attribute) can be value-restricted to a subconcept of the range inherited from its superconcept. In KOLA, the range of a necessary condition can be an interval, a set of numbers, or a single number as well as a concept: for example, the interval **[29 50]** or the set **{2 4 7}** [3]. The classifier determines subsumption relationships using the subset

[3]Their syntax is accounted for in **Appendix** A.1.

relationship between ranges.

As an example, consider **Figure** 5.2 (a). The concept *Domain-2* is defined as a subconcept of the concept *Domain-1*. In addition to just inheriting roles or attributes in the concept *Domain-1*, the role *role* (the attribute *attribute*) in the concept *Domain-2* can be value-restricted from the concept *range-1* to the concept *range-2*, where the concept *range-2* is a subconcept of the concept *range-1*.

Unlike in KANDOR, the range of a role in a concept in KOLA must be a concept and cannot be an assertional value.

5. Number Restriction:

   Just as in KL-ONE, we specify the cardinality information about role fillers plausible in a concept and define a new concept by restricting the number of role fillers. Conventially, the cardinality of the set of role fillers is specified in the form (l, u), as shown in **Figure** 5.2 (b): when a concept is instantiated, the number of fillers of this role is at least l and at most u. Similarly, we can impose the number restriction on the fillers of an attribute in a concept.

   The subsumption relation by the number restriction is determined by the subset relation of intervals. To define the concept *Domain-2* as a subconcept of the concept *Domain-1* by the number restriction, the following condition has to be satisfied:

$$l_1 \leq l_2 \text{ and } u_1 \geq u_2$$

6. Differentiation of a necessary condition:

   A new concept can be defined by differentiating a necessary condition. Differentiation allows the specification of a subrole by restricting its range

to a subconcept of the range of its superrole. A subrole is to be filled with subsets of the fillers of the role it differentiates.

7. Attribute to role restriction:

   We can define a concept as a subconcept of other concepts by changing the status of a necessary condition from an attribute to a role. For example, suppose *gender* is not a definitional necessary condition for an object to be a person and, thus, *gender* is defined as the attribute for the concept *Person*. When we define the concept *Male-Person* as a subconcept of the concept *Person*, we have to mention *gender* to describe a male person. Thus, *gender* becomes the role in the concept *Male-Person*. This is called "attribute to role" restriction in KOLA. See (c) in **Figure** 5.2

8. Restriction of ranges related by a *Part-of*-like necessary condition:

   A concept can defined from more general concepts by restricting the range of an inherited role (attribute) to a concept related to the range inherited from the superconcept by a *Part-of*-like necessary condition. This restriction effectively imposes a range restriction on the inherited necessary condition.

Formally, let A_d, B_d, A_r, and B_r be concepts. Suppose the structures of A_d and B_d are the same except for the range of role (attribute) **RA**: the range of **RA** in A_d is A_r, while that of **RA** in B_d is B_r. We need to prove that if A_r and B_r are related by a *Part-Of*-like necessary condition, say **PA**, then B_d is a subconcept of A_d. Consider the following:

Suppose B_d were not a subconcept of A_d. In other words, there exists an instance of B_d which is not an instance of A_d. For any instance I_{Bd} of B_d, there exists a filler I_{Br} of **RA**, where I_{Br} is an instance of B_r. Since

A_r and B_r are related by the *Part-Of*-like role PA, in I_{Br}, there exists PA's filler I_{Ar} which corresponds to I_{Bd}. Thus, I_{Bd} muse also be an instance of A_d, which contradicts the assumption that B_d was not a subconcept of A_d.

As an example, we can define *Nephrotic-Disease* as a subconcept of *Kidney-Disease* by restricting the range *Kidney* of the role *Anat-Involvement* to *Nephron* which is related to *Kidney* by the role *Part-of*. Thus, any nephrotic disease is also a kidney disease.

9. Role/Assertional Constraints.

KOLA uses role constraints to define new concepts. A role constraint represents a relationship between the sets of fillers of roles in that concept [Moser 83]. In KOLA, the constraints definitional on necessary conditions are considered as role constraints, while the constraints on necessary conditions which can be suppressed are called assertional constraints.

Role or assertional constraints are represented by a chain of necessary conditions. In order to represent the constraints on necessary conditions, the graphical notation of KL-ONE is adopted with the addition of a label (r/a) inside the diamond which indicates whether it is a role constraint or an assertional constraint.

As an example, consider how a person with the same disease as his mother can be defined. Suppose, from the concept *Patient* which is defined as an example in Appendix A.4, we define the concept *Patient-with-Patient-Mom* in which the role *Mother* is value restricted to the concept *Patient*. Now, we can define the concept *Patient-with-Mom's-Disease* as the subconcept of the concept *Patient-with-Patient-Mom* by using the following role constraint: disease of the patient is the same as that of his

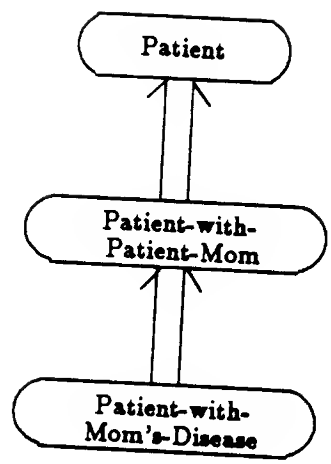
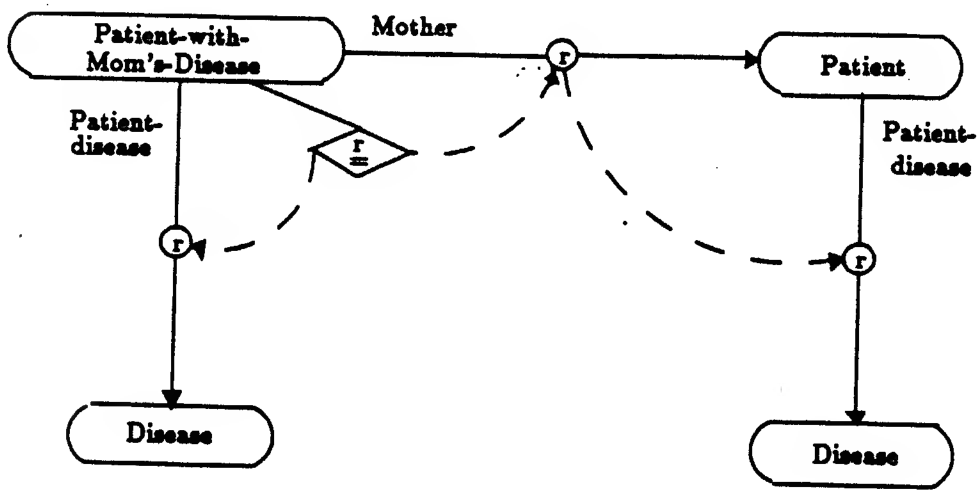


Figure 5.3: Subsumption Relationship among Patient and its subconcepts

mother. Figure 5.3 shows the subsumption relationship between the concepts of concern. Figure 5.4 shows the constraints on necessary conditions that an instance must satisfy to be a member of the concept *Patient-with-Mom's-Disease.*



(Patient-Disease (Patient-with-Mom's-Disease))
= (Patient-Disease (Mother (Patient-with-Mom's-Disease)))

Figure 5.4: Example of a role constraint

In KOLA, five operators are provided to represent the constraints on necessary conditions: equal, greater than or equal to, less than or equal to, greater than, and less than. Actually, only three operators – less (greater), less than (greater than), and equal – are needed, and the rest of them can be achieved with transposing two chains in a constraint.

The core of C-World is the classifier. Major functions of the classifier are building the concept taxonomy (and role taxonomy) based on the subsumption

relationship and performing the categorical, terminological reasoning about concepts or instances.

When a concept is defined, the classifier places it between its most specific superconcepts and most general subconcepts. In KOLA, each concept keeps information only on its most specific superconcepts and its most general subconcepts. Using this information and the transitive property of subsumption relationship, the classifier is able to perform inference about the subsumption relations between concepts efficiently.

5.1.3 Role/Attribute Properties

Transitivity

From the discussion in `Section` 4.2.1, we recognize that every (at least) necessary condition used to define a concept might not have unique characteristics [4].

In previous concept-based knowledge representation systems, there is no distinction between a role with the transitivity property and a role without it, let alone a facility to deal appropriately with such a distinction. Thus, the queries *1), Is the nephron a part of the urinary system?* and *2), Is the kidney anatomically involved in Brian's nephrotic disease?* cannot be answered correctly. These kinds of questions, which are related to a role with transitivity,

[4] Such a characteristic is one that is applicable to any necessary condition of a concept regardless of its definitionality. Thus, though only roles are mentioned, the discussion can also be applied to attributes.

require more than a single step of search to reach a correct answer, and must be handled by a problem-solving system external to a knowledge base [5].

Consider the role *Anatomical-Part-of* involved in query 1. The nephron is an anatomical part of the kidney, and the kidney is an anatomical part of the urinary system. From this knowledge, we can infer new knowledge that the nephron is an anatomical part of the urinary system. The role *Anatomical-Part-of* is transitive. Suppose, on the other hand, that the concept *Male-Person* has the role *gender* whose role filler's concept is the concept *Genders* for representing the class of possible genders. The role *gender* is not transitive.

We need a facility to help the system to distinguish *Anatomical-Part-of*-like roles from *gender*-like roles and deal with them appropriately. The system needs to be told that roles can be divided into two classes: ones with transitivity and ones without it. The knowledge that a role is transitive need be given to a system explicitly, so that it can be used in subsequent reasoning processes.

The following is the definition of transitivity of a necessary condition.

Definition :

Let c_1, c_2, and c_3 be concepts each of which has a (at least) necessary condition $\mathcal{AR}$. Let $c_i\mathcal{AR}c_j$, mean that the range of $\mathcal{AR}$ in the concept c_i is the concept c_j. $\mathcal{AR}$ is transitive if and only if $c_1\mathcal{AR}c_2$ and $c_2\mathcal{AR}c_3$

[5] As an aside, recall a concept-based knowledge representation system's major property. certain kinds of inferences can be directly based on the structure of a concept-based knowledge base, and the inferential operations can be reduced to a simple graph search of some sort. This allows a concept-based knowledge representation system to have high performance.

implies $c_1 \mathcal{AR} c_3$, where $C_i \neq C_j$, if $i \neq j$, for $i, j = 1, 2, 3$.

The language construct *Transitive* in KOLA tells a system about a necessary condition's transitivity.

Observe that there are queries which require the further transitive search through another necessary condition, after a search through the one directly related to them being performed. Consider query *2* in **Section** 4. To find the anatomical involvement of Brian's nephrotic disease, the role *Anatomical-involvement* in the concept *Nephrotic-D* will be searched, and its range reached. Our interest, however, is not whether *Nephron* is the anatomical involvement of Brian's disease, but whether *Kidney* is. To arrive at a correct answer, we must traverse the network through the role *Anatomical-Part-of.* In other words, answering this question requires further search through the role *Anatomical-Part-of* of the concept *Nephron*. The language construct *IndirectTransitive* in KOLA is the construct to tell a system about such knowledge. This construct is described in detail in **Section** 5.3.

Let us consider how transitivity of a *Part-of*-like necessary condition, say *PA*, affects classification. Suppose that for a concept A_d, which has a role *Role*, whose range is A_r, we define a new concept B_d as a subconcept of A_d by value-restricting *Role* to a concept B_r, where *Role* is indirectly transitive through PA. If the subsumption relationship between A_r and B_r is not specified explicitly, the classifier in KL-ONE establishes the subsumption relationship between A_r and $B_r \wedge A_r$, as described in **Section** 4.2.1. The classifier in KOLA, however, tries to figure out how A_r and B_r are related to each other. If it finds that they are related by PA, the classifier does not establish the subsumption relationship

between them.

For example, reconsider the example from Section 4.2. While KL-ONE's classifier constructs a new concept $KIDNEY \wedge NEPHRON$ and classifies it as a subconcept of the concept *KIDNEY*, KOLA's classifier does not make such a new conjoined concept, since *KIDNEY* and *NEPHRON* are related by the transitive role *Part-of* (see Section 5.1.2).

In addition, if both A_d and B_d are defined as subconcepts of SUP, the classifier in KOLA establishes the subsumption relationship between A_d and B_d, if it finds that A_r and B_r are related by PA. For example, suppose the domain is modeled as follows:

```
(defrole PART-OF primitive
     (domain ANAT-PART)  (range ANAT-PART))
(defrole ANAT-INVOLVEMENT primitive
     (domain DISEASE) (range ANAT-PART))

(defconcept KIDNEY primitive (specializes ANAT-PART))
(defconcept NEPHRON primitive (specializes ANAT-PART)
    (res PART-OF (vrc KIDNEY)))
(defconcept KIDNEY-DISEASE primitive (specializes DISEASE)
    (res ANAT-INVOLVEMENT (vrc KIDNEY)))
(defconcept NEPHROTIC-DISEASE primitive (specializes DISEASE)
    (res ANAT-INVOLVEMENT (vrc NEPHRON)))
```

Given the fact that *PART-OF* is transitive and *ANAT-INVOLVEMENT* is indirectly transitive via *PART-OF*, the classifier finds that both *NEPHROTIC-DISEASE* and *KIDNEY-DISEASE* have the indirectly transitive role *ANAT-*

INVOLVEMENT and their ranges, *KIDNEY* and *NEPHRON* are related by the transitive role *PART-OF*. Using this information, the classifier classifies the concept *NEPHROTIC-DISEASE* as a subconcept of the concept *KIDNEY-DISEASE*.

Let us consider another example.

```
(defrole PART-OF primitive
     (domain ANAT-PART)  (range ANAT-PART))
(defrole ANAT-INVOLVEMENT primitive
     (domain DISEASE) (range ANAT-PART))

(defconcept KIDNEY primitive (specializes ANAT-PART))
(defconcept NEPHRON primitive (specializes ANAT-PART)
    (res PART-OF (vrc KIDNEY)))
(defconcept PROXIMAL-NEPHRON (specializes NEPHRON))

(defconcept KIDNEY-DISEASE primitive (specializes DISEASE)
    (res ANAT-INVOLVEMENT (vrc KIDNEY)))
(defconcept NEPHROTIC-DISEASE primitive (specializes DISEASE)
    (res ANAT-INVOLVEMENT (vrc NEPHRON)))
(defconcept PROXIMAL-NEPHROTIC-DISEASE primitive
    (specializes NEPHROTIC-DISEASE)
    (res ANAT-INVOLVEMENT (vrc PROXIMAL-NEPHRON)))
```

NEPHRON has the role *PART-OF* whose range is *KIDNEY*, and *PROXIMAL-NEPHRON* is a subconcept of *NEPHRON*. Thus, *PROXIMAL-NEPHRON* in-

herits all necessary conditions of *NEPHRON*, including *PART-OF*. *PART-OF* of *PROXIMAL-NEPHRON* is also value-restricted to *KIDNEY*. Consequently, along with information about indirect transitivity of *ANAT-INVOLVEMENT*, the system can determine whether or not *KIDNEY* is *ANAT-INVOLVEMENT* of *PROXIMAL*-NEPHROTIC-*DISEASE*. More generally, let A and B be concepts where A is *PART-OF* B. If we define A_{SUB} as a subconcept of A, then A_{SUB} is *PART-OF* B because of inheritance.

Symmetry and Inverse Relation

Symmetry is another property which necessary conditions of a concept can have.

In KOLA, when it is specified that a concept A has a necessary condition, say NEC, whose range is a concept B, declaring the symmetry of NEC allows the classifier to add NEC whose range is A into the set of necessary conditions of B. This may cause the problem with the circularity which is described in `Chapter` 6.

Declaring the symmetry of a necessary condition, `NC`, tells the system the following. Let `INS` be an instance of a concept which has the necessary condition `NC`. If the range concept also has `NC` as one of its necessary conditions and `NC`'s filler in `INS`, say `FILLER`, is specified, then `NC` in `FILLER` has `INS` as its filler.

Consider the example in `Appendix` A.4. The *Spouse* is declared as a symmetric role. In the example, the instance *Jason* has *spouse* whose filler is *Mary*, but the instance *Mary* does not have any information about her spouse. However, from knowledge that the role *Spouse* is symmetric, the system can infer

that the instance *Mary* has the property [6] *Spouse* whose filler is *Jason*. KOLA uses the symmetry of a necessary condition to represent knowledge directly. In previous concept-based knowledge representation systems, some inferential operations are required to get information about Mary's spouse because the description about *Mary* does not have any information about her spouse. In KOLA, however, by telling the system about the symmetric property of the necessary condition *Spouse*, knowledge about Mary's spouse is represented vividly. Thus, inferential operations to draw information about Mary's spouse are reduced to a simple retrieval operation.

Some necessary conditions have the inverse relation: for example, the necessary conditions *Contains* and *Contained-In*. Information about the inverse relation between necessary conditions is manipulated by KOLA in a way similar to symmetry. In other words, such information is used to represent knowledge vividly.

Vivid representation of such knowledge about instances influences the instantiator to establish instantiation links between instances and concepts, and ultimately terminological reasoning about membership of an instance. Details are given in **Section** 5.2.2.

In summary, the classifier memorizes information about the symmetry of a necessary condition and inverse relation between necessary conditions. Such information allows the instantiator in I-World and Question-Answerer to work

[6] A property in an instance will be accounted for in **Section** 5.2.1.

correctly and efficiently. Details are covered in **Section** 5.4.

The way to get to answers

There is often more than one way to reach the same place in a domain. For example, to get to the Symphony Hall in Boston, there are several choices for transportation – foot, bus, subway, or taxi. In KOLA, this idea is adopted in solving a problem. Consider the example in **Appendix** A.4. Even though the instance *Mary* does not have any information about her children, we can infer that both Kib and Brian are her children, because she is Jason's wife and Jason has Kib and Brian as his children. The system needs to know that children can be computed indirectly by using *Spouse* and *Children*. KOLA uses the language construct *prop-assert* to tell the system knowledge about how to get to an answer. The following examples show how we can tell the system such knowledge:

```
(prop-assert children ::= spouse children)
(prop-assert mother ::= father spouse)
(prop-assert father ::= mother spouse)
```

One's children can be reached by searching one's spouse's children, one's mother by searching one's father's spouse, and so on.

The classifier memorizes information on the way to get to an answer so that Question-Answerer can perform correctly and efficiently.

5.1.4 Knowledge about Terminological Knowledge

In addition to terminological knowledge for modeling objects in a domain, knowledge about terminological knowledge(KTK) may be available. Some KTKs are used to perform reasoning about the domain efficiently. Some KTKs affect the structures of concepts.

KTK in KOLA includes disjointness, cover, and synonyms of a concept. The classifier memorizes disjointness, cover, and synonyms of concepts and uses them in building the concept taxonomy. Question-Answerer uses KTK in C-World to solve a problem efficiently. I will focus on how they are dealt with in KOLA, because they were already described in KL-ONE,

Disjointness and Cover

A disjointness class is a set of concepts which cannot have any common instances. Thus, given a disjointness class of concepts, if an instance is a member of one concept, then it cannot be an instance of any other concept. For example, suppose we define three concepts, ***short-person***, ***average-person***, and ***tall-person*** by value-restricting the role corresponding to a person's height appropriately. We can define them as a disjointness class: a short person cannot be a tall one at the same time. Knowledge of disjointness can be valuable in reasoning. For example, the disjointness class was used as a potential tool in ABEL's diagnostic reasoning [Patil].

We may accidentally define a concept which is a subconcept of both the concept *Short-person* and the concept *Tall-Person*, although we have defined

them as the disjointness class. Incoherencies caused by the violation of disjointness classes should be checked for. There are at least three ways possible to handle incoherencies of disjointness classes.

1. *Careful Checking Strategy*:

   The classifier considers information on disjointness classes, while performing classification. If the classifier detects incoherency in a disjointness class while classifying concepts, it stops classifying concepts. Unfortunately, this strategy can be computationally expensive. Checking each newly defined concept would require $\mathcal{O}(n^2)$ disjointness checks, where n is the number of superconcepts above the new concept. Moreover, disjointness may be defined after the disjoint concepts and combinations of them have already been defined, once again requiring $\mathcal{O}(n^2)$ disjointness checks, where n is the number of concepts in the hierarchy below the disjointed concept.

2. *Postponed Checking Strategy*:

   The classifier does not use information on disjointness classes: instead, the instantiator uses it Checking the violation of disjointness classes is postponed until an instance is created. Because a disjointness class is a set of concepts for which there are no common instances, this method is reasonable if no instance in the domain exists. However, this strategy is also costly because whenever an instance is created, it must check for a disjointness class violations.

3. *Never-Mind Strategy*: Incoherency due to violated disjointness class is totally ignored. Checking to see the violation of a disjointness class is not performed when the concept taxonomy is built or when an instance

is created based on the assumption that such violation cannot happen. If we choose this strategy, we have to accept any problem which may result from such an assumption.

In KOLA, the *Careful Checking Strategy* and the *Never-Mind Strategy* are employed. A knowledge engineer can choose either of them based on the characteristics of a domain of concern. If the violation of a disjointness class can cause unacceptable problems in subsequent reasoning processes, take the *Careful Checking Strategy*. If such a violation is acceptable, the *Never-Mind Strategy* can be chosen.

A concept is said to be covered by a set of concepts, called a *covering set*, when it is exhausted by the concepts in the covering set. Every instance in a covered concept will also be an instance of at least one of concepts in the covering set.

Synonyms

There are many notions which can be referred to by more than one term in a domain of concern. For example, in the medical domain, a disease or a symptom can be denoted by several terms: `Jaundice` [7] and `icterus` refer to the same symptom, and `Granulomatous lymphoma` and `Hudgkin's disease` denote the same disease. `Deoxyribonucleic Acid` can be denoted by its abbreviation *DNA*.

[7] Jaundice is a yellowing of the skin and whites of the eyes, indicating excess bilirubin (a bile pigment) in the blood [Med-Dictionary].

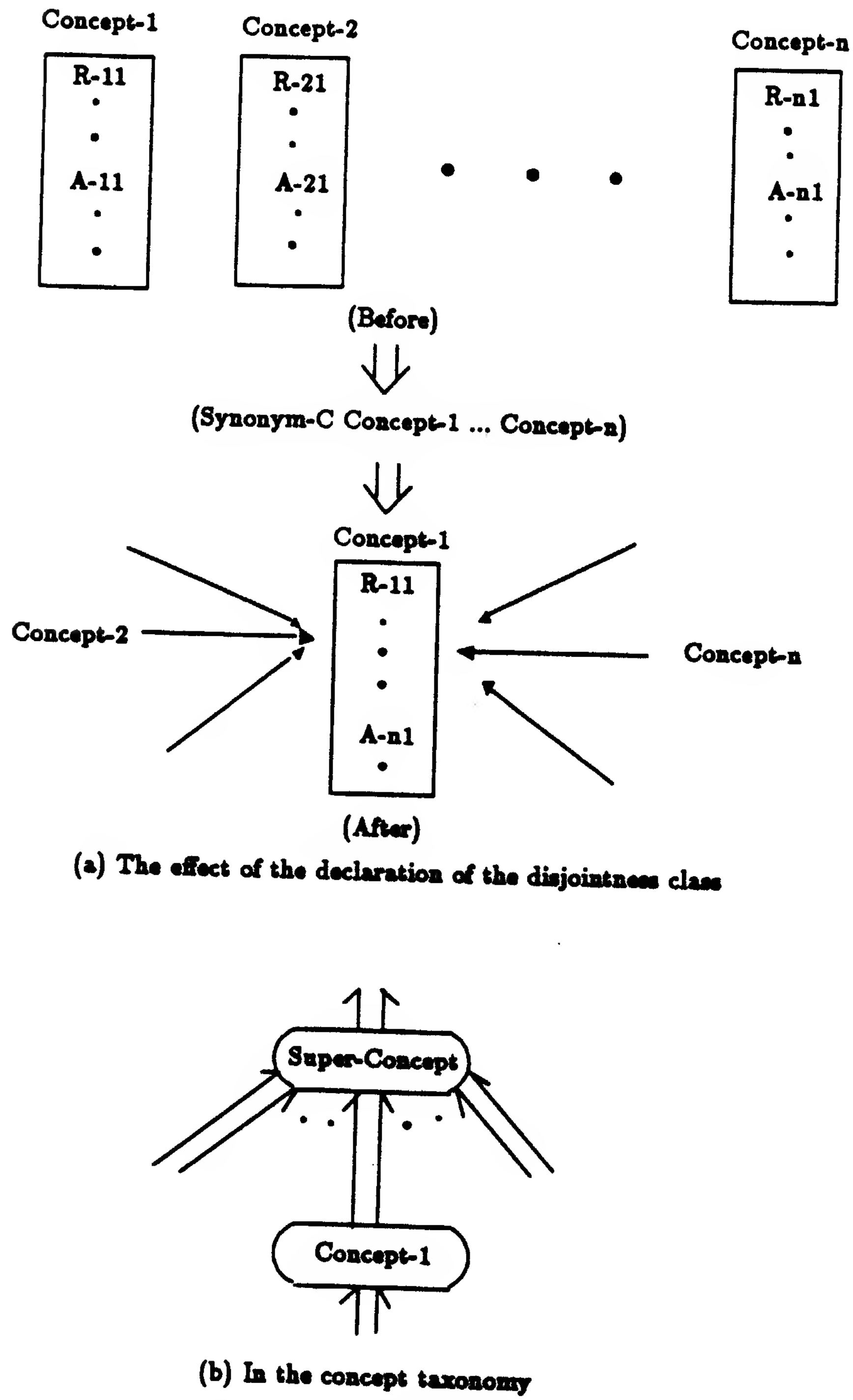


Figure 5.5: The effect of the declaration of synonyms

Information on synonyms of terms is valuable to the classifier in KOLA. When information about synonyms of concepts is given, the classifier merges all the information, such as necessary conditions, which are spreaded over several concepts into one structure. The classifier makes one representative for the concepts each of which defines the same entity. Only the representative is placed into the concept taxonomy. All concepts except the representative in the declaration of a synonym are called *dummy* concepts in KOLA. Later, if information on a concept is given via a dummy, the classifier puts it into the corresponding representative. Figure 5.5 shows how KOLA deals with the declaration of synonyms. To represent the structure of a concept, a rectangle is used. After a declaration of synonyms, the structures of all concepts included in this declaration are merged into one structure, and dummy concepts point to the concept *Concept-1* which is designated as the representative. The representative concept *Concept-1* is placed in the concept taxonomy, while the others are not.

In KOLA, it is recommended that the declaration of synonyms be done before the classification of concepts starts. If information on synonyms is given after the concept taxonomy is built, the classifier has to merge the structures of the concepts included in the synonym declaration into one structure, and propagate the merged structure to their subconcepts along the concept taxonomy. Thus, a late declaration of synonyms disturbs the concept taxonomy. Moreover, the propagation of the effects of synonym declaration is computationally expensive: it would require $\mathcal{O}(n^2)$, where n is the number of subconcepts of all the concepts included in the declaration of synonyms.

5.1.5 C-World Utility

In KOLA, the classification of concepts [8] is performed by the function (`classification` [*option*]). The result of the function `classification` is the concept taxonomy based on the subsumption relationship and concepts' structures which are interrelated to each other appropriately. When classification fails, the reasons for the failure are related to a user. (Details are found in `Appendix` A.7.

After classification, a user can ask about a concept. KOLA's response is a properly indented description in which important information is written in bold face. A description of a concept consists of three parts. The first part of the description gives the user the perspective on the concept, by presenting brief information about it. In the second part, more detailed information about the concept is shown. In this part, KOLA gives definitional information of the concept, and queries if a user wants to see nondefinitional information about the concept. The third part contains information which differentiate the concept from its superconcepts. This part is also included in the description only on demand.

As an example, consider `Figure` 5.6 which is the output of the command `Show Concept` on *Symbolics 3650* lisp machine [9]. It is the description of the term *Serum-HCO3-Concentration* which is generated by KOLA. A concept is represented by its name with the prefix |C|, a role by its names with the prefix

[8] It is covered in `Appendix` A.3 how the classifier performs the classification of concepts.

[9] Details are found in `Appendix`.

```
Command:  Show Concept (a concept name) Serum-HCO3-Concentration

    Let us see the concept |C|SERUM-HCO3-CONCENTRATION

    * Brief Description:
      Its superconcepts: |C|SERUM-PARAMETER
      Its definitional necessary conditions:
       |R|:VALUE, |R|MEASURED-FROM, |R|MEASURED-SEX, |R|PARAMETER-OF,
      |R|MEASURE-OF

    * Details:
      - Primitive and Generic
       - Immediate Super Concepts:
           |C|SERUM-PARAMETER
      - Immediate Sub Concepts:
           |C|ARTERIAL-SERUM-HCO3-CONCENTRATION
           |C|VENIOUS-SERUM-HCO3-CONCENTRATION

      - Roles and their restrictions:
       |R|:VALUE :
            - Value-restricted to [21 29]
            - No number restriction.
         |R|MEASURED-FROM :
            - value-restricted to |C|ANAT-ENTITY
            - No number restriction.
         |R|MEASURED-SEX :
            - value-restricted to |C|GENDERS
            - No number restriction.
         |R|PARAMETER-OF :
            - value-restricted to |C|SERUM
            - No number restriction.
         |R|MEASURE-OF :
            - value-restricted to |C|HCO3
            - No number restriction.

  * Do you want to see its difference from its superconcepts? (Yes or No) Yes
    Differences
     |R|:VALUE: Val-Restriction
      |R|MEASURE-OF: Val-Restriction
```

Figure 5.6: Description of the concept Serum-HCO3-Concentration

```
Command: Show Concept (a concept name) person

    Let us see the concept |C|PERSON

    * Brief Description:
       Its definitional necessary conditions:
         |R|MOTHER |R|FATHER |R|CHILDREN

    * Details:
     - Primitive and Generic
     - Defined as the concept in the top level of the concept taxonomy
     - Immediate Sub Concepts:
         |C|MALE-PERSON |C|FEMALE-PERSON |C|MARRIED-PERSON |C|DOCTORS
         |C|PATIENT

    - Roles and their restrictions:
        |R|MOTHER :
          -  value restricted to |C|MARRIED-FEMALE-PERSON
          -  the size of the set of its fillers is exactly 1
        |R|FATHER :
          -  value restricted to |C|MARRIED-MALE-PERSON
          -  the size of the set of its fillers is exactly 1
        |R|CHILDREN :
          -  value restricted to |C|PERSON
          -  the size of the set of its fillers is at least 1


    *  Do you want to see its nondefinitional necessary conditions? (Yes or No) Yes

    -  Attributes and their restrictions:
        |A|OCCUPATION :
          -  value restricted to |C|JOB
          -  the size of the set of its fillers is at least 1
        |A|ANATOMY :
          -  value restricted to |C|ANAT-ENTITY
          -  the size of the set of its fillers is exactly 1
        |A|AGE :
          -  value restricted to |C|NUMBERS
          -  the size of the set of its fillers is exactly 1
        |A|GENDER :
          -  value restricted to |C|GENDERS
          -  the size of the set of its fillers is exactly 1
        |A|NAME :
          -  value restricted to |C|STRINGS
          -  the size of the set of its fillers is at least 0

** MORE **
```

Figure 5.7: Description of the concept Person

Command: Show Taxonomy (a concept name) Serum-Parameter

Super Concepts **Sub Concepts**

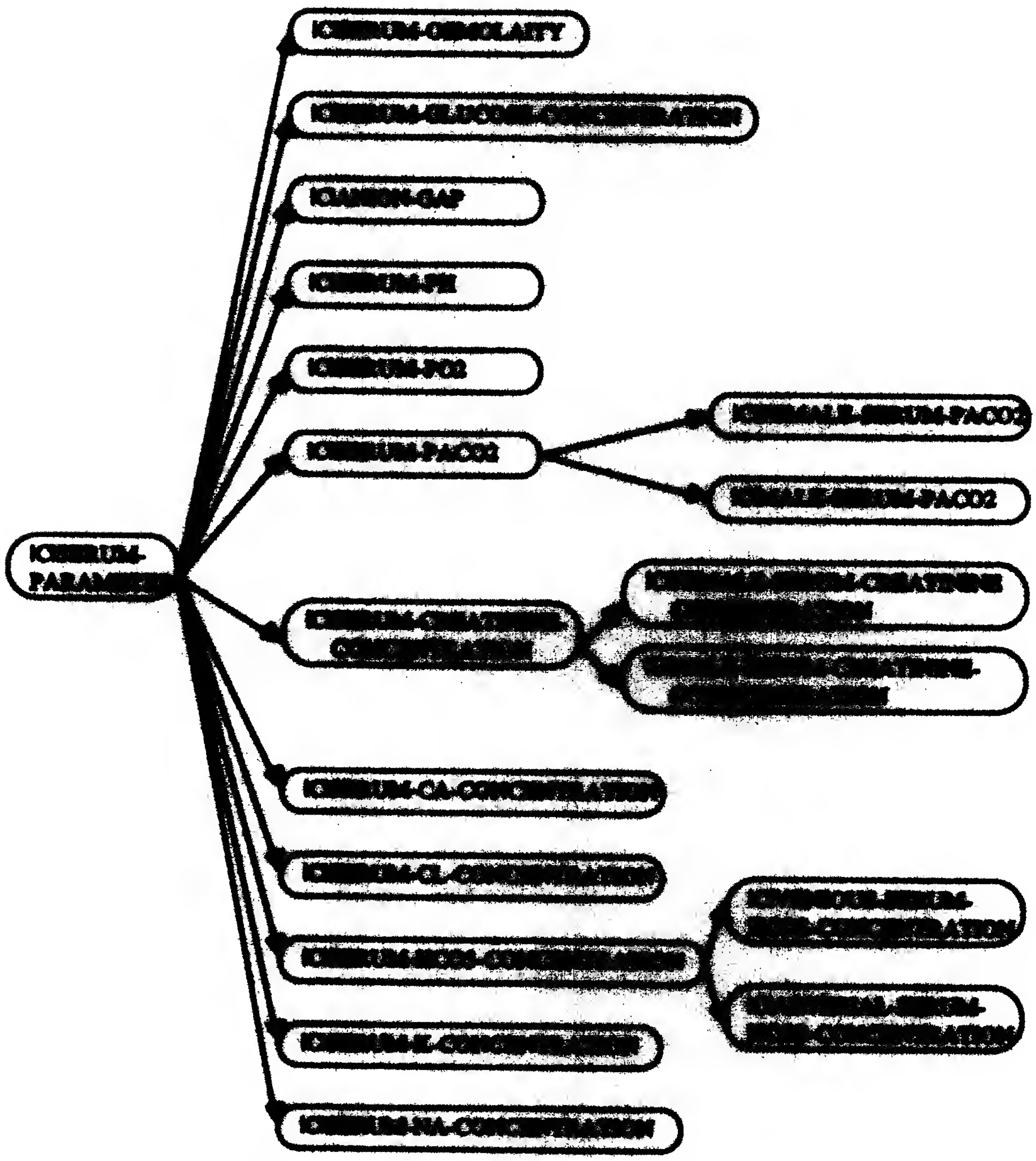


Figure 5.8: Concept Taxonomy whose root is ICSERUM-PARAMETER

|R|, and an attribute by its name with the prefix |A|. From the brief description, we know that it is the subconcept of the concept *Serum-Parameter* and has several roles, including *:Value*.

More detailed information about the concept is given in the second part. The second part contains information about value- and number-restriction of each role, and information about role constraints.

`Figure` 5.7 is the description of the concept *Person*, and shows how non-definitional information about a concept is included in KOLA's description.

KOLA also has the ability to graphically display information on the subsumption relationship between concepts. `Figure` 5.8 is the output of the command `Show Taxonomy` whose root is the concept *Serum-Parameter*, and shows the subsumption relationship between the concept *Serum-Parameter* and its subconcepts. (Only three levels of the taxonomy are explored because of the limited size of the window.)

5.2 I-World

I-World consists of assertions. In KOLA, such assertions are made by the instantiation of concepts in C-World. In addition, I-World contains knowledge about assertions, such as Detailed filler references and synonyms of instances, that helps Question-Answerer reason about assertions.

5.2.1 Instances

KOLA is told contingent facts about its domain as instances. An instance is an existing object in the domain, and is made by the instantiation of a concept. Such an instance is said to belong to the instantiated concept. An instance is assertional in nature. KOLA deals with an instance in a way similar to how KANDOR deals with an individual. Unlike KL-ONE, KOLA does not treat an instance as a kind of concepts. An instance is given to a system by means of a description which consists of at least one of the following:

- A set of concepts each of which the instance must belong to.
- A set of properties which the instance has in a domain.

The structure of an instance is determined by the concepts whose instantiation it is. An instance inherits all necessary conditions from all concepts instantiated to it. Unlike the creation of a concept, no operations, except simply inheriting necessary conditions from its instantiated concepts, are allowed when an instance is created. In other words, none of the operations discussed in Section 5.1.2 are allowed when an instance is created.

In KOLA, when an instance is created, a necessary condition inherited from its concept becomes a *property* of the instance. A property in an instance is represented by its name and its fillers. If a property is explicitly mentioned in the description of an instance, each of its fillers must be an instance and cannot be a concept, because value restrictions are not allowed when an instance is created. No instance can have as its property a property which is not defined as a necessary condition in C-World

When an instance, `INS1`, is created, it can contain a property, `PROP`, which is not specified explicitly in the description of `INS1` but is included in the set of properties inherited from its concepts. Suppose that no filler of `PROP` in `INS1` is given. KOLA deals with such an instance as follows. First, KOLA tries to find out if `PROP`'s fillers can be found indirectly using information such as symmetry or inverse relation.

- When no filler still is found:

  even when there is the minimum number restriction to `PROP`, `INS1` can be created in KOLA. In KANDOR, the creation of such an instance is treated as an inconsistency. KOLA accepts the lack of information about its property when an instance is created. Thus, when the description of an instance is asked for later, the property without fillers can be included in the returned description: instead, the concept which each of its potential fillers belongs to is mentioned.

- When fillers are found, use them as the fillers of `PROP` in `INS1`.

When an instance is created, instance-specific information on the cardinality of fillers may be known, regardless of their exact identity in a domain. There are three ways to tell the system about the number of fillers, when an instance is created in KOLA:

- a property's fillers specified in the description are all the fillers the property can have in the domain. Such information is given to the system by means of the keyword `:all`. For example, if `(Prop :all)` is included in the description of an instance where `Prop` is this instance's property, then the

fillers which have been specified so far for the property `Prop` are all the fillers that its property `Prop` can have in the domain.

- though all elements in a set of fillers are not known, the total number of fillers can be known when an instance is created. The expression `(:all x)` in KOLA is the facility to tell the system that the total number of fillers for a property is exactly `x` where `x` is a non-negative integer.
- although the total number of fillers is not known, it can be known that there will be more fillers than specified explicitly in the domain. The keyword `:canbemore` is used to tell the system about such fact.

In order to deal with a property which does not have any information on the cardinality of its fillers, the closed world assumption is adopted as default in reasoning about an instance: in other words, for such a property, it is assumed that its fillers specified in the description of an instance are all the fillers it has. In the question-answering process, a user is informed if an answer is obtained based on the closed world assumption.

Pictorial notations of instances and the relationships between them are provided with KOLA. An instance in KOLA is denoted by a rectangle which contains the name of the instance. A property is denoted by a single arrow which relates an instance to either other instances or a concept which, when no filler is specified yet, a potential filler must belong to. `Figure` 5.1 (d) shows both that an instance *Instance* has the property *Property* and that its property *Property*'s fillers are not specified yet in a domain, but, if they are given later, all of them must be instances for the concept *Range-Concept*. `Figure` 5.1 (e) shows that the instance *Instance* is related to the instance *Filler* by the property

Property. A property in an instance can have more than one filler. `Figure` 5.1 (f) shows that the property *Property* in the instance *Instance* has $n(\geq 2)$ fillers each of which is known to the system.

Instance Network

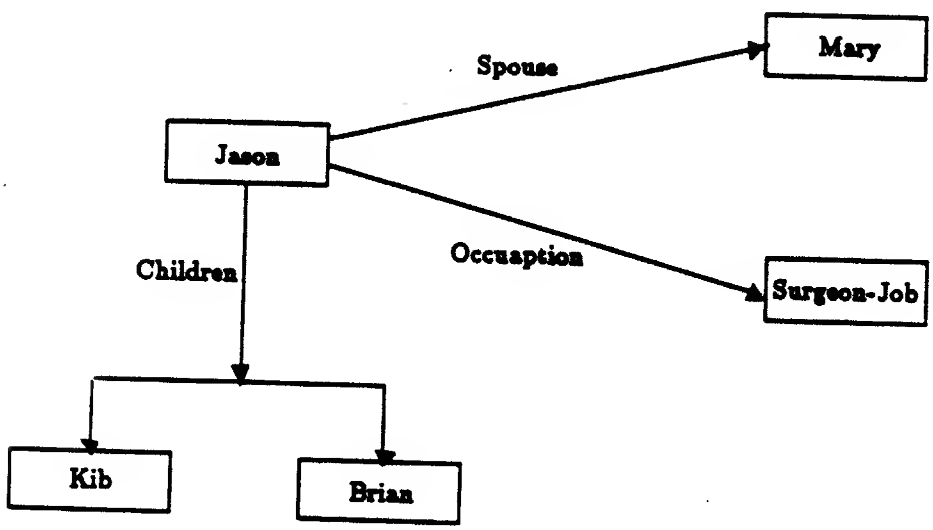


Figure 5.9: Example of the Instance Network

As instances are created, an instance network is created in I-World. The instance network is the network which is built based on instances and relationships among them. A node in the instance network is an instance. A branch is a property

which relates one instance to other instances in a domain. The instance network shows how instances are related to each other, and thus reflects the structure of the domain. **Figure** 5.9 shows part of the instance network built by the example in **Appendix** A.4: it shows how the instances *Jason*, *Mary*, *Kib*, *Brian*, and *Surgeon-Job* are related to each other.

5.2.2 Instantiator

The instantiator is the major component of I-World. The instantiator not only builds the *instance network* which captures the structure of its domain but also establishes the *instantiation link* which connects an instance to the concepts it belongs to.

Whenever an instance is created, the instantiator determines the instance's structure. The instantiator figures out the structure of an instance based on description. As mentioned in **Section** 5.2.1, the description of an instance consists of a set of concepts each of which the instance must belong to and/or a set of pairs each of which consists of a property and its filler(s). Based on its structure, the instantiator links the instance to the most appropriate concepts to which it belongs. Even if no concept which an instance must belong to is given in its description, the instantiator has the ability to find the concepts to which the instance belongs by manipulating given properties and fillers. Terminological reasoning about an instance can be done by searching through an instantiation link.

Given an instance INS, the instantiator finds the most appropriate con-

cepts which INS belongs to. Let CON be one of such concepts. Then, CON satisfies all of the following:

1. If concepts are specified in the description of INS as instantiated concepts, CON is a subconcept of at least one of these concepts. Otherwise, CON has as its necessary conditions at least one property which is found in the description of INS.

2. For all properties found in both the set of CON's necessary conditions and in the set of properties in INS's description, no fillers of any of such properties violate any restrictions, such as value- and number-, or constraints.

3. If CON is not a subconcept of any concept specified in the description of INS, it must be non-primitive. (In order for an instance to be the instantiation of a primitive concept, the fact that it is an instance of the primitive concept has to be explicitly specified in the instance's description.)

An instantiation link which connects an instance with its most specific concept(s) is pictorially represented by a single arrow with an encircled *i* as shown in Figure 5.1 (c). The classifier provides for the instantiator all of information necessary to determine the concepts of an instance.

For example, consider the following which is the description of the instance *Jason* from Section 4:

```
(definstance Jason (:Instanceof person)
    (name "Jason Lee")
    (children (:all 2) Kib Brian)
    (gender Male)
```

```
(spouse Mary)
(occupation :canbemore surgeon-job)
(age 40))
```

The syntax of `definstance` is covered in **Appendix** A.1. Jason who is an instance of the concept *Person* has six properties explicitly specified such as *name*, *children*, *etc.* The expression **(:all 2)** informs the system that for the instance *Jason*, the total number of fillers of the property *Children* is exactly 2. The keyword `:canbemore` denotes that, even though we don't know exactly what they are, we know that a property can have more fillers than specified explicitly in the description. Jason not only practices medicine but may also have other jobs.

Based on the properties *gender* and *spouse* and their fillers' adherence to given restrictions or constraints, the instantiator can conclude that the instance *Jason* can also be the instance of the concept *Married-Male-Person* as well as the concept *Person* (See **Appendix** A.4 for the complete knowledge base under consideration). Thus, the instantiator connects the instance *Jason* with the most appropriate concept *Married-Male-Person*, because it is the most general concept which include *gender* and *spouse* in necessary conditions. Even though the instantiation link between the instance *Jason* and the concept *Person* is not established explicitly, the system can make an inference that *Jason* is the instance for the concept *Person* using the transitive property of the subsumption relationship. *Jason* is the instance for the concept *Married-Male-Person*, and the concept *Married-Male-Person* is the subconcept of the concept *Person*. Therefore, *Jason* is the instance for the concept *Person*.

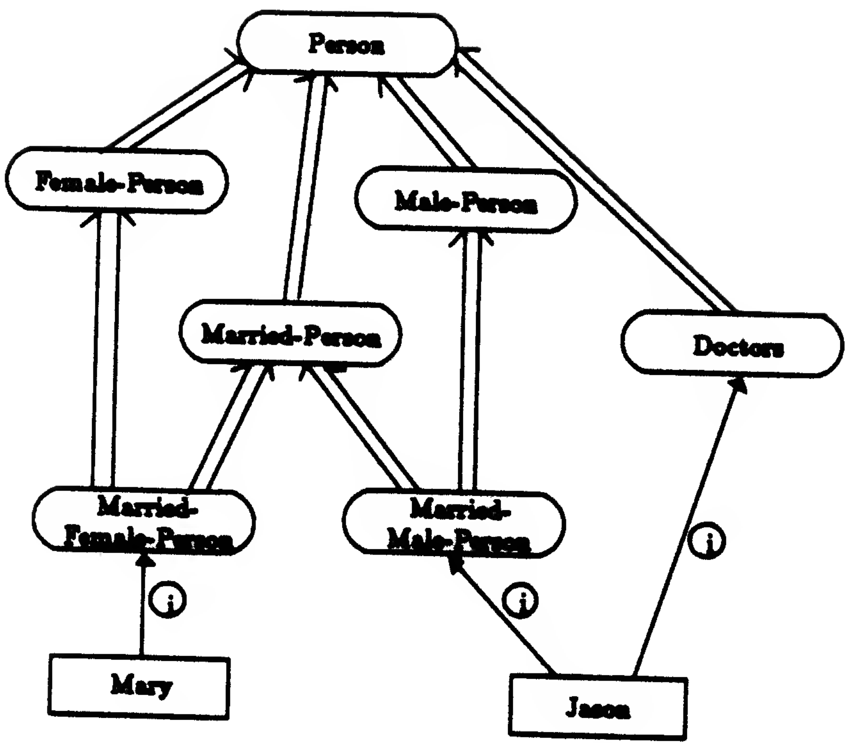


Figure 5.10: Example of the instantiation links between instances and their concepts

In addition, the instantiator connects the instance *Jason* with the concept *Doctor* based on the property *occupation* and its filler. The instantiator always tries to connect an instance with the most appropriate concepts whose instantiation it is, and thus with the concepts *Married-Male-Person* and *Doctor* instead of the concept *Person* as shown in Figure 5.10.

Although Mary is said to be the instance of the concept *Person*, the instantiator can connect the instance *Mary* with *Married-Female-Person* by manipulating the given description about her. This example shows how vivid representation of knowledge affects instantiation. If, during instantiation, the fact about Mary's spouse is not inferred from the given facts and represented vividly, the instance *Mary* is connected at most to the concept *Female-Person*.

The connection of an instance with its most appropriate concepts is not static in KOLA. Though an instance was already connected to some concepts, the instantiator detaches it from the currently connected concepts, and connects it with more specific concepts to which it belongs, as more information about this instance is gathered. An instance does not stay at the initially connected concept, but is dynamically linked with more specific concepts. This is valuable in reasoning about whether or not an instance is the instantiation of a concept.

5.2.3 Detailed Filler References

Consider an existing concept-based knowledge representation system. After an instance is created, we can ask about the fillers of a role that we are interested in, and the answer is obtained by searching through the role in the instance. When a concept is instantiated, we can pre-specify the number of fillers of a

role which can possibly be filled by means of number restrictions. However, it is the whole set of fillers that can be named and referred to. Each (or part) of the fillers of a role cannot be easily named or used in subsequent courses of solving a problem, even though we know that there is a convenient, economical way to reach them. In previous concept-based knowledge representation systems, there has not been an efficient way to let the system know that there is such a convenient way to get to the particular filler, although we know that.

One possible solution for this problem is to differentiate the role. However, this solution may not be desirable because such information needed to reach a particular filler from a particular instance is instance-specific. Therefore, differentiating a role makes the structure of a concept unnecessarily fat as discussed in `Section` 4.2.2.

In `Section` 4.2.2, we discussed the inefficiency caused by using role differentiation or simple role inheritance to solve queries such as query 3, *"Who is Jason's first son?"*. All of the methods suggested turned out to be inefficient. They make the role taxonomy messier, and require more memory.

In the example in `Section` 4.2.2, Jason has two children: Kib and Brain. We can easily tell the system these facts by simply filling the property *Children* in the instance *Jason*.

Now, suppose that it is known that Kib is the first son of Jason in our domain. Should there be a way to tell the system this fact efficiently? Even though the first son may not be generic enough to be defined as a necessary condition of a person, it would be better if the system could be told that Kib

can be reached from Jason by using the first son as a kind of reference.

The assertional ability of KOLA can be improved, by telling the system that some of the fillers of a property in a particular instance can be reached through a particular reference and by letting it use such knowledge in subsequent problem-solving sessions. This facility is provided by the *detailed filler references* in KOLA.

The following is another use of the detailed filler references. A domain may require that fillers of a property should be represented with some particular structure when an instance for a concept is created. The system should be told the structure for such potential fillers. For example, it may sometimes be useful to store given fillers in order of age when an instance for the concept *Person* is created.

The common property of specific reference to a particular filler and structure within multiple fillers is that such information is likely to be clarified after an instance is created, and is not enough to be defined as a necessary condition of a concept.

In KOLA, information about the possible structure within potential fillers of a necessary condition, **NEC**, can be specified in a concept. This information is simply handled just as inherited data of **NEC**, not as a necessary condition. Even though such information is inherited by subconcepts of the concept, it does not affect the classification of concepts because it is not a necessary condition.

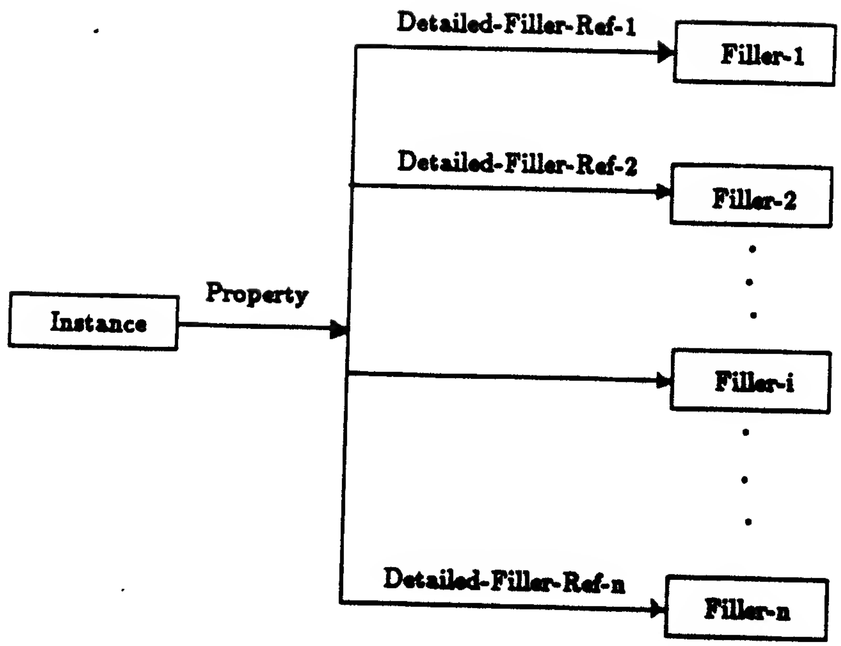
When such information is specified in a concept, we need to decide when to use such information. Do we use this information and store the filler within

the prespecified structure, whenever a necessary condition's filler is given? This is not efficient because, supposing the concept has hundreds of subconcepts, whenever any of them is instantiated, fillers of a property with this information would have to be stored within the prespecified structure even when it is unnecessary. In KOLA, when `NEC`'s filler is given, it is manipulated in the same way as a filler without such information is manipulated. KOLA uses information about the possible structure within potential fillers to answer them, only when questions which require the use of such information are asked.

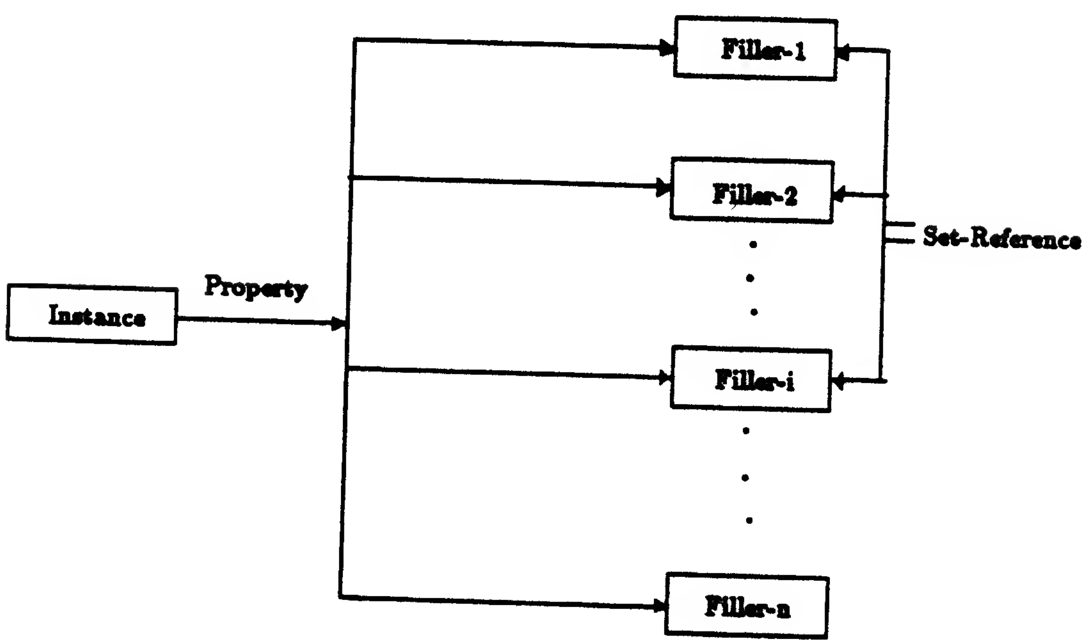
The ***detailed filler references*** enable us to express what might be represented in a knowledge base more succinctly and easily, and allow the system to answer queries more efficiently. The construct for ***detailed filler references*** is described in `Section` 5.3 along with its semantics.

In KOLA, detailed filler references of a property do not play any role in determining the role taxonomy. Only the role or attribute related to detailed filler references participates in constructing the role taxonomy as a representative. The goal of detailing a property is to facilitate to access some fillers in a property in a particular instance.

Detailed filler references can also be represented in the instance network. Consider `Figure` 5.11. `Figure` 5.11 (a) shows the pictorial representation of a detailed filler reference in the instance network. `Figure` 5.11 (b) shows the graphical representation of such a situation in the instance network. `Set-Reference` with the double line in `Figure` 5.11 (b) shows that `Filler-1`, `Filler-2`, and `Filler-i` form a set and can be reached by `Set-Reference`.



(a) Detailed Filler References for an individual



(b) Detailed Filler References for a group

Figure 5.11: Pictorial Representation of Detailed Filler References

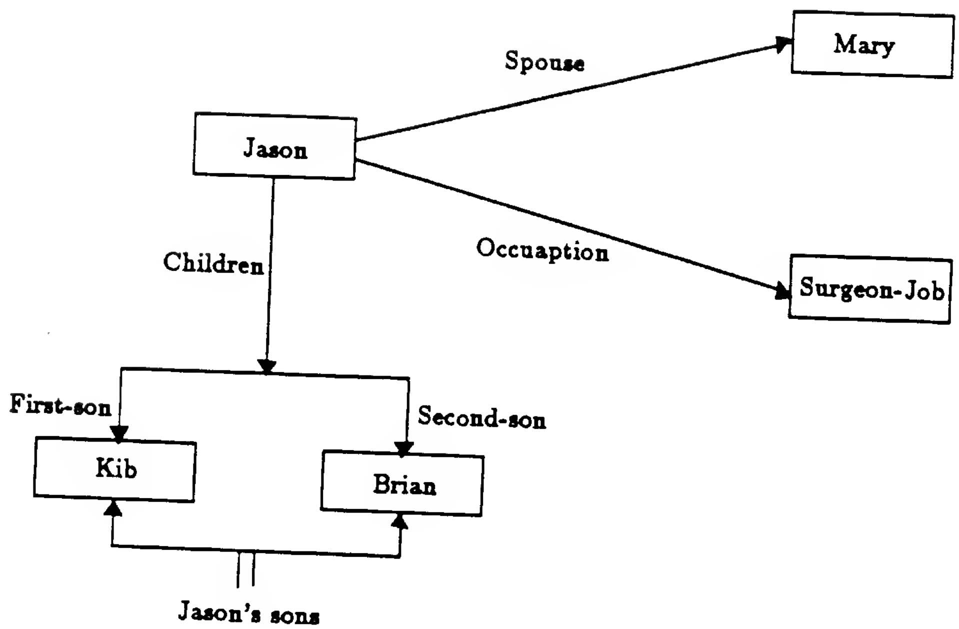


Figure 5.12: Example of the instance network with detailed filler references

`Figure` 5.12 is the augmented instance network of `Figure` 5.9 with detailed filler references: Kib can be reached by `First-son`, and Brian by `Second-son`. If it is known in the domain that Kib and Brian are Jason's sons, the system can be told this fact with the detailed filler reference `Jason's sons`. `Jason's sons` with the double line in `Figure` 5.12 shows that Kib and Brian are Jason's sons, and Jason's sons can be reached by the detailed filler reference `Jason's sons`.

5.2.4 Synonyms for instances

Like a term, an instance can be referred to by several names. For example, suppose Kib and Jason's first son are defined as instances in our domain:

```
(definstance Kib ....)
(definstance Jasons-First-son ....)
```

Suppose Kib and Jason's first son denote the same person in the domain. The system needs to be told that these two instances represent the same entity. Like concept synonyms, only one structure is created as the representative for instances specified in the synonym declaration. Other dummy instances point to this representative. (The transparency is another issue, and can be handled at the reasoning level, not at the representation level. The facility to handle this problem is not implemented in the current version of KOLA.)

5.2.5 I-World Utility

In KOLA, instantiation of an instance is performed by the function `(instantiation)`. (Details are found in **Appendix** A.7.) Before performing the main stage of the instantiation, the instantiator vividly represents knowledge which otherwise would be implicit, based on the symmetry and inverse relation of necessary conditions. The result of the function `instantiation` is the instantiation links between instances and concepts, and the instance network which captures the structure of the instantiated domain. When the instantiation fails, the classifier gives the user the reasons for the failure.

After instantiation, a user can ask questions about the description of an instance. KOLA answers by giving an appropriately indented description in which important information is written in bold face. **Figure** 5.13 shows the output of the command **Show Instance** for the instances *Jason* and *Mary*.

5.3 Semantics for Transitivity and Detailed filler References

There are two important operators incorporated with a knowledge base – the *ASK* operator for asking questions of the knowledge base and the *TELL* operator for telling the knowledge base new knowledge [Newell 81]. Telling a system about the transitive property of a role or attribute, or detailed filler references helps the system infer implicit knowledge efficiently.

```
Command: (instantiation)
       Done........

Command: Show Instance (a instance name) Jason

    Let us see the instance |I|JASON
    - Explicitly specified concepts whose instances it is:
        |C|PERSON
    - Its concepts obtained by inferential operations:
        |C|DOCTORS |C|MARRIED-MALE-PERSON

    * Do you want to see fillers of roles or attributes |I|JASON has? (Yes or No) Yes

      |P|CHILDREN: |I|KIB |I|BRIAN
      |P|OCCUPATION: |I|SURGEON
      |P|GENDER: |I|MALE
      |P|SPOUSE: |I|MARY

      |P|NAME: Jason Lee
      |P|AGE: 40


Command: Show Instance (a instance name) Mary

    Let us see the instance |I|MARY
    - Explicitly specified concepts whose instances it is:
        |C|PERSON
    - Its concepts obtained by inferential operations:
        |C|MARRIED-FEMALE-PERSON

    * Do you want to see fillers of roles or attributes |I|MARY has? (Yes or No) Yes

      |P|GENDER: |I|FEMALE
      |P|SPOUSE: |I|JASON

      |P|NAME: Mary Lee
      |P|AGE: 35
```

Figure 5.13: Descriptions of the instances Jason and Mary

The following describes the semantics of the transitivity and *detailed filler references* constructs in KOLA. To define the semantics of each of them, the formalism used in [Schomolze 83] is adopted. That is, M denotes the following:

> A semantics for a KL-ONE network will be given in a standard first order language with lambda abstraction (called FOL+). With some network N, we associate a set of predicates, one predicate corresponding to each element of N_k, and a set of sentences in FOL+, one sentence corresponding to each element of N_p. $\cdots$
>
> $\cdots$ M takes each element of N_k into a (possibly complex) predicate, which is denoted in FOL+.
>
> – *Classification in the* KL-ONE *knowledge representation system* [Schomolze 83]

The semantics of the constructs is presented in terms of question-answering problems (subsequently referred to as QA problem). There are two types of QA problems – *Is*-questions which ask about existence and *wh*-questions which ask about what it is as well as existence. In subsequent formulas, we use the bound variable f to indicate whether a question under consideration is a *Is*-question or a *Wh*-question.

5.3.1 Transitivity

In this section, the construct for representing the transitivity of a role or an attribute in KOLA is explained in detail.

$$\begin{aligned}
(M\ (\mathit{Trans}\ AR)) = \lambda_{fx}.[\exists \mathcal{Z} = \{z_1, z_2, \ldots, z_{n-1}, z_n\}] \\
\wedge\ [(AR \notin ARs(z_n)) \vee (z_1 = z_{n+1}, where\ \ (M, AR)z_n z_{n+1})] \\
\wedge\ [z_i \neq z_j, if\ \ i \neq j, 1 \leq i, j \leq n] \\
\wedge\ [(M, AR)xz_1 \wedge (M, AR)z_1 z_2 \wedge \cdots \wedge (M, AR)z_{n-2} z_{n-1}
\end{aligned}$$

$$\wedge(M, AR)z_{n-1}z_n]$$
$$f = is \rightarrow (\ \lambda_y.\ y \in \mathcal{Z} \rightarrow T,\ error),\ \mathcal{Z}$$

Information about role/attribute transitivity allows Question-Answerer to efficiently infer knowledge that is implicit in a knowledge base. Let AR denote a role or an attribute, and x, $k, y, z_1, \ldots, z_n$, and z_{n+1} denote concepts. $ARs(z_n)$ denotes the set of all necessary conditions that a concept z_n has. Given the concept x, if 1) x has the necessary condition AR whose range is the concept z_1; 2) each concept z_i has the necessary condition AR whose range is the concept z_{i+1} for $i = 1, \cdots, n-1$; 3) z_n does not have AR as its necessary condition, or $z_1 = z_{n+1}$ where z_n has AR as its necessary condition whose range is z_{n+1}; 4) $z_i \neq z_j$, if $i \neq j$ for $i, j = 1, \cdots, n$; and 5) y is one of z_i's, then the answer is *Yes* in *Is*-query, while the answer should be the set of z_i's in *Wh*-query.

Among the five conditions, the condition 3) specifies how the classifier deals with a transitive necessary condition with circularity. The classifier keeps track of concepts searched through AR. If it detects that a concept under consideration was already visited, the classifier stops searching and gives the answer.

With this construct, we can represent the knowledge given in `Section 4` as follows:

```
(Transitive anatomical-part-of)

(defconcept urinary-system p (s anatomical-entity))
(defconcept kidney p (s anatomical-entity)
            (role anatomical-part-of (vrc urinary-system)))
(defconcept nephron p (s anatomical-entity-by-function)
```

(role anatomical-part-of (vrc kidney)))

For simplicity, the details of other concepts such as *anatomical-entity*, or *anatomical-entity-by-function* and their relationships are not included [10]. To understand how transitive property of the role *anatomical-part-of* is used, consider the query 1 again. The query *1, "Is the nephron a part of the urinary system?"* is a *is*-query which has *Anatomical-Part-of* as a role . The values of bound variables in the lambda expression for this query are as follows: f's value is *is*, x's value is *nephron*, y's value is *urinary-system*. Because the set Z is {kidney, urinary-system} and y whose current value is *urinary-system* is in Z, the system will reach the answer *Yes*.

Consider the following lambda expression for the construct to tell a system that further search through another role or attribute is required to answer queries such as question *2*.

$$
\begin{aligned}
&(M\ (IndirTrans\ AR\ \texttt{:via}\quad AR_{imp}))\\
&= \lambda_{fx}[\exists k, (M, AR)xk]\\
&\qquad f = is \rightarrow (\lambda_y.\ k \neq y \rightarrow [\exists Z = \{z_1, \ldots, z_n\}]\\
&\qquad\qquad \wedge\ [(AR_{imp} \notin ARs(z_n)) \vee (z_1 = z_{n+1}, where\quad (M, AR_{imp})z_n z_{n+1})]\\
&\qquad\qquad \wedge\ [z_i \neq z_j, if\quad i \neq j, 1 \leq i, j \leq n]\\
&\qquad\qquad \wedge\ [(M, AR_{imp})xz_1 \wedge (M, AR_{imp})z_1 z_2 \wedge \cdots \wedge (M, AR_{imp})z_{n-1}z_n]\\
&\qquad\qquad y \in Z \rightarrow T,\ error),\ k
\end{aligned}
$$

This expression is similar to the one for *Transitivity*. The system needs to know that, because Brian has a nephrotic disease, he has a disease in the

[10] The anatomical knowledge base for urinary system has been built as a running example for demonstrating KOLA's capability.

kidney that the nephron is a part of. The system is told this fact through `IndirTrans`. In this case, given `(indirTrans anatomical-involvement :via anatomical-part-of)`, f's value is *is*, x's is *nephrotic-disease*, y's is *kidney*, AR is *anatomical-involvement*, AR_{imp} is *anatomical-part-of*, and k's is *nephron*. The system will find that *kidney* $\neq$ *nephron* and, thus, continue its search via *Anatomical-part-of*. It will conclude that Z's value is {kidney} and reach the answer *Yes*.

5.3.2 Detailed Filler References

The construct for detailed filler references provides a way of telling the system about information which is not essential, but helps the system reason about implicit information more efficiently. Consider the following:

$$
\begin{aligned}
&(M\,(DetailFR\ Ins\ Prop{:}FR\ F)\\
&= \lambda_{fx}.[Prop \in INSProp(Ins) \wedge F \subset Fillers \wedge FR \in DetailedFRs(Prop)]\\
&\qquad \rightarrow f = is \rightarrow (\lambda_y.\ x = Ins \wedge y = F \rightarrow T,\ error),\\
&\qquad\qquad (x = Ins \rightarrow F, error)\\
&\qquad error
\end{aligned}
$$

Ins is an instance , *Prop* a property in *Ins*, FR a detailed filler reference available in a domain, and F is a set of instances. $INSProp(Ins)$ is a function to compute all properties the instance *Ins* has. *Fillers* is the set of all known fillers of the property *Prop* in the instance *Ins*. $DetailedFRs(Prop)$ is the function to find the set of all detailed filler references which were already known to reach *Prop*'s fillers individually or by group. By means of the construct for the detailed filler references, the system can be told that the instance *Ins* has the property *Prop*, and F among its fillers can be reached through the detailed

reference *FR*. Question-Answerer can use this information, when solving a problem.

Using the example from `Appendix` A.4, KOLA can be told that Kib is the first son of Jason as follows:

```
(DetailFR (In Jason) Children:First (Fillers Kib))
```

`in` and `Fillers` are the keywords in KOLA to represent that the property *Children* in the instance *Jason* is involved and that *Kib* can be referred to through the detailed filler *First-son*, respectively.

5.4 Question-Answerer

There is no guarantee that all knowledge in a domain is explicitly represented in a description about the world . Thus, a knowledge representation system should have inferential capability to draw new conclusions about its world by manipulating knowledge represented explicitly.

The assertional capability of the existing concept-based knowledge representation systems is limited to allowing a user to assert statements of existence, establishing statements of coreference of descriptions, and making statements of identity of individual constructs in a particular situation. Consequently, improvements in the ability to draw what is implicit from assertional knowledge represented explicitly is very desirable.

In KOLA, Question-Answerer is responsible for performing *ASK* opera-

tions. With the support of Question-Answerer, KOLA has the ability to identify a concept or an instance in its domain. We can ask Question-Answerer questions such as

- given properties each of which is given with or without its fillers, find an instance or instances satisfying them,
- given necessary conditions each of which is given with or without restrictions or constraints, find a concept or concepts satisfying them, or
- given properties each of which may or may not accompany fillers, find a concept or concepts which an instance (or instances), which satisfies given conditions, belongs to.

In addition to facilities provided by previous concept-based knowledge representation systems, KOLA can also answer questions requesting information on a property in a particular instance or on a particular necessary condition in a concept. For the question, ***Who are Mary's children?***, Kib and Brian should be returned as the answer because Mary is Jason's wife. For the question, ***Who is Mary's first son?***, Kib should be returned. Such questions are asking for implicit knowledge and, thus, requires inferential operations. In KOLA, Question-Answerer has the ability to deal with such questions efficiently. Question-Answerer has two operators for this type of *ASK* operations: one for *Is*-questions and one for *Wh*-questions.

- *Is*-operator:

  We may be interested in knowing if a property in an instance has another instance as its filler, or if a concept has a necessary condition whose

```
Command: (Is Kib (children first-son) :of Mary)
         Yes.

Command: (Is Mary children :of Jason)
         Definitely No, because all fillers of |P|CHILDREN in |I|JASON are
          known, and |I|MARY is none of them.

Command: (Is Jason spouse :of Mary)
         Yes.

Command: (Is "James" name :of Jason)
         No (by closed world assumption).

Command: (Is professor occupation :of Jason)
         Maybe. Altough it is not explicitly known that PROFESSOR is
         a filler of |P|OCCUPATION, it is known that |I|JASON can have
         more fillers in |P|OCCUPATION.

Command: (Whatis :ins occupation :of :ins Mary)
         Sorry. I cannot find the answer you want. I, however, know that
         fillers of |P|OCCUPATION have to be the instance of |C|JOB.

Command: (Whatis :c :value :of :c rectal-temperature)
         37.4

Command: (Whatis :c :value :of :c Female-serum-Paco2)
         [32 45]

Command: (Whatis :ins occupation :of :ins Jason)
         |I|SURGEON

Command: (Whatis :ins (children second-son) :of :ins Mary)
         |I|BRIAN
```

Figure 5.14: Example of ASK operations

value is another concept. KOLA can be asked such questions with the following:

(Is *Range Nec* :of *Domain*) [11]

For example, we can ask if Jason's occupation is Surgeon:

(Is *Jason Spouse* :of *Mary*).

For this type of question, Question-Answerer tries to find as correct an answer as possible. First, Question-Answerer attempts to determine whether *Domain* is a concept or an instance. If *Domain* is a concept, and it has *Nec* as one of its (at least) necessary conditions, Question-Answerer finds the range of *Nec*. If the range found is equivalent to *Range*, Question-Answerer will return *Yes*. Otherwise, it will returns *No*. In the process of finding the corresponding range, Question-Answerer can use knowledge about terminological knowledge in C-World such as knowledge about a role's transitivity or concept synonyms.

If Question-Answerer finds out that *Domain* is an instance, it will try to find fillers of the property *Nec* in the instance *Domain*. If the instance *Range* is in the set of the fillers found, then *Yes* will be returned. *No* or *Maybe* will be returned, otherwise. In the case in which the answer is *No* or *Maybe*, Question-Answerer tries to give a user the reason for the negative answer, based on information such as cardinality or the closed world assumption. Figure 5.14 shows answers of KOLA for several queries.

When the system fails to determine if *Domain* is a concept or an instance, it asks a user to give more information about *Domain*'s identity.

[11]Its syntax is described in Appendix A.6.

In order to reach an answer, Question-Answerer uses assertional knowledge such as detailed filler references, synonyms, or cardinality of fillers.

- *Wh* operator:

  We may want to know

  - what is the range of a necessary condition of a concept,
  - what are the fillers of a property of an instance in a domain , or
  - what is the value restriction on a property in an instance.

  The following is the operator in KOLA to ask such questions:

  (`Whatis` *flag1 Nec* `of` *flag2 Domain*) [12]

  flag1 and *flag2* are the flags to tell what our interest is, an instance or a concept. They are either `:C` or `:Ins`. If both *flag1* and *flag2* are `:Ins`, Question-Answerer will return the fillers of the property *Nec* in the instance *Domain*. If both *flag1* and *flag2* are `:C`, then the range of the necessary condition *Nec* in the concept *Domain* will be returned. If *flag1* is `:C` but *flag2* is *:Ins*, Question-Answerer will find the range which each of fillers of the property *Nec* in the instance *Domain* belongs to. Any other combinations of *flag1* and *flag2* is illegal in KOLA. In the process of finding an answer, if Question-Answerer needs to use knowledge about terminological knowledge or about assertional knowledge, it comes into contact with C-World or I-World to get necessary information.

`Figure` 5.15 shows how three subsystems in KOLA interact with each other. I-World uses C-World to determine the instance's structure and to refine

[12]Its syntax is covered in **Appendix** A.6.

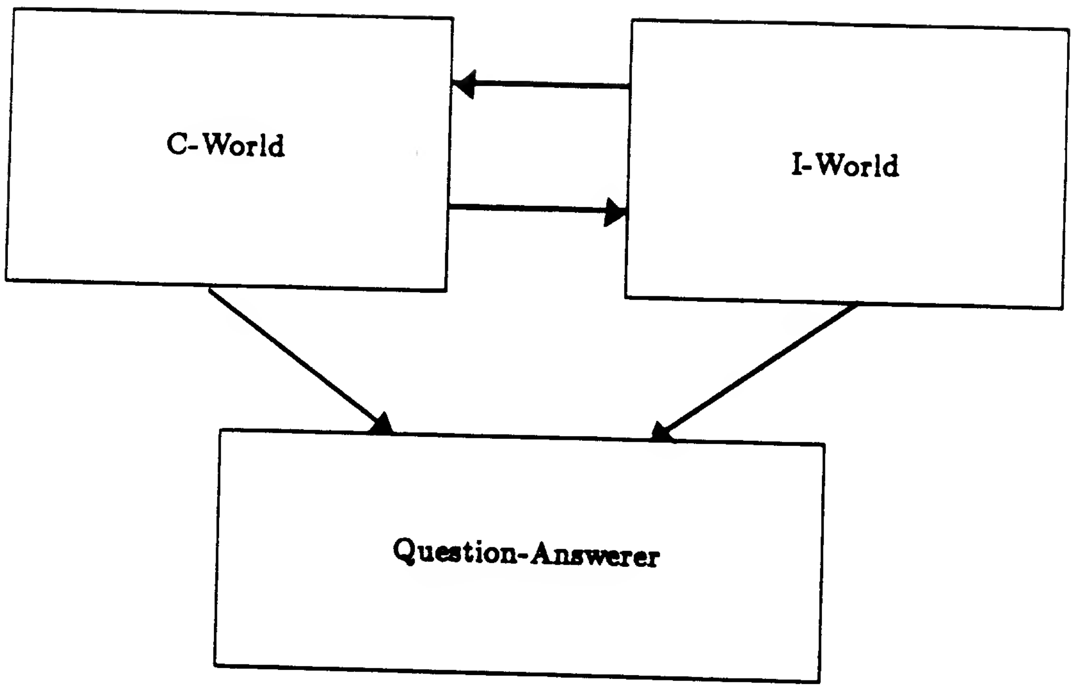


Figure 5.15: Subsystems of KOLA and Relationship among them

the instance network whenever an instance is created. C-World uses information in I-World to know which instances are connected to which concepts. Whenever an instance is created, it is connected with the most appropriate concepts which it belongs to. Question-Answerer tries to use not only terminological knowledge in C-World and assertional knowledge in I-World, but also knowledge about such knowledge to answer questions as correctly and efficiently as possible.

Chapter 6

Conclusion

Making an intelligent system depends largely on the method used for storing, retrieving, and manipulating knowledge. To do this, embedding knowledge within a computer system is indispensable. The selection of a knowledge representation formalism to model a domain is critical yet risky, because we cannot, at the time of selection, know exactly the quality or variety of knowledge. It may turn out that important knowledge cannot be represented easily after most part of the knowledge base has already been completed. Throughout this paper, we have shown how a knowledge representation system such as KOLA, which is able to capture the structure of a domain, can be used to model a wider variety of knowledge than previous systems.

A good solution of a problem can be achieved with a good choice of knowledge representation systems. A good solution is an acceptable one that is reached efficiently within a reasonable amount of time. A concept-based knowledge representation system attempts to embed the structure of a domain in its

computational model, under the assumption that the structural representation of knowledge will facilitate reasoning capabilities of the system. KOLA is implemented as a concept-based knowledge representation system.

KOLA achieves a new state of the art in knowledge representation with facilities which improve expressiveness and reasoning ability. KOLA attempts to represent knowledge as succinctly and vividly as possible, using distinction between definitional and nondefinitional necessary conditions of a concept, explicit declaration of types of relations among concepts, and detailed fillers references. As a result, some inferential operations are reduced to simple retrieval, which allows KOLA to accomplish improved reasoning performance.

6.1 Future Work

KOLA still has the difficulties with multiple definitions of a concept, *OR* concepts, and structural descriptions.

KOLA cannot deal with a concept with multiple definitions. Multiple definitions of a concept cause undecidability in the process of building the concept taxonomy, and thus are not allowed in a concept-based knowledge representation system. However, if multiple definitions of a concept were available, they would be of great value because a single term can sometimes be defined in more than one way [1].

[1] A single notion can be defined in more than one way even in the same domain. For example, in the medical domain, the term *acidemia* may be defined either as a decreased PH or as an increase in hydrogen ion concentration.

Similarly, KOLA cannot handle a concept defined by disjointing existing concepts. Even though *OR* concepts without structures are nicely manipulated in [Haase 87], further work needs to be done to handle *OR* concepts with their own structures.

The problem with structural descriptions [2] is determining if one structural description is subsumed by another structural description, when trying to classify a concept. We want to be able to say things like:

- the concept *Urgent-Message* is defined from the concept *Message* [Brachman 85] as *a message with a reply time of less than one hour.*

Role value maps allow set/subset relations among fillers, but they are not general enough. For example, we cannot say that "the volume of a solution = the volume of the solvents in the solution" without using structural descriptions.

Finally, KOLA does not handle the circular definitions of concepts appropriately. For example,

(defconcept A **(Role** R_1 **(Vrc** B**)))**
(defconcept B **(Role** R_2 **(Vrc** A**)))**

Classification of A is not independent of that of B, because classification of A requires classification of B, and vice versa. In KOLA, suppose A is classified before B is classified. When A is classified, B is treated as a concept with only those superconcepts which are explicitly presented in its description. Classification based on this assumption is flawed. For example, suppose that C is a

[2] Structural descriptions specify how the components in concepts are related to each other.

concept with R_1 whose range is D, and that B is a subconcept of D, but D is not one of superconcepts declared explicitly in B's description. We want A to be classified as a subconcept of C. However, when A is classified, B is not classified, and it is assumed that B is not a subconcept of D. Thus, the classifier cannot establish the subsumption link from A to C. We need a facility to deal appropriately with the dependencies caused by such circularity.

Appendix A

Appendix

A.1 Syntax for Concepts and Instances

KOLA ::= KOLADefinition +
KOLADefinition ::= ConceptDefinition|RoleDefinition|AttributeDefinition|InstanceDefinition
ConceptDefinition ::= (**Defconcept**|**Defc** Conceptname {Conceptdescription})
ConceptDescription ::= Specializationconcepts|PrimitiveSpec|InvidualSpec|
RestrictionForm|ConstraintForm|AttachedIData|AttachedData
Specializationconcepts ::= (**S**|**Specializes** Conceptname +)
PrimitiveSpec ::= **P**|**Primitive**
IndividualSpec ::= **I**|**Individual**
RestrictionForm ::= (**Role**|**Attribute** Rolename|Attributename RestrictionSpec) +
RestrictionSpec ::= ValResrtic|NumRestric +
ValRestric ::= (**Vrc**|**Vrconcept** conceptname|interval|numberset)
NumRestric ::= (**Num**|**Number** integer)|(**Max** integer)|(**Min** Integer)|
(**Max** integer) (**Min** integer)|(**Min** integer) (**Max** integer)
ConstraintForm ::= (**Role**|**Assertional** nec-Chain operator nec-Chain) +
nec-Chain ::= (nec +)
nec ::= Rolename|Attributename
operator ::= != | !>= | !<= | !< | !>
AttachedIData ::= (**Idata** values)
AttachedData ::= (**Data** values)
values ::= number | string | symbol | list
Conceptname ::= symbol
Interval ::= [lower-bound upper-bound]

numberset ::= { $num_1 \cdots num_n$ }
lower-bound ::= number
upper-bound ::= number
num_i ::= number, $i = 1, \ldots, n$
Rolename ::= `:value`|symbol
Attributename ::= symbol
RoleDefinition ::= (`defrole`|`defr` Rolename [Role-specs])
AttributeDefinition ::= (`defattribute`|`defa` Attribute [Attribute-specs])
Role-specs ::= Spec +
Attribute-specs ::= Spec +
Spec ::= (`domain` Conceptname)|(`range` Conceptname)|
    (`number` number)|(`max` number)|(`min` number)|
    (`differentiate`|`diff` $RA_1 \cdots RA_n$)|
    (`idata` lisp-object)|(`data` lisp-object)

InstanceDefinition ::= (`Definstance`|`Defins` instancename InstanceSpec)
InstantiationSpec ::= InstantiationSpec|PropertySpec
InstantiationSpec ::= (`:instanceOf` conceptnames)
conceptnames ::= conceptname +
PropertySpec ::= (Propertyname [NumSpec] FillersSpec) +
NumSpec ::= `:all` | (`:all` number) | `:canbemore`
FillersSpec ::= instancename +
Propertyname ::= symbol
instancename ::= symbol

$< NB >$:

- number, string, symbol, and list are lisp objects.
- [and] in [lower-bound upper-bound] are the reserved keywords to tell the system about an interval. The blank between [and *lower-bound* (between *upper-bound* and]) is imperative. Also, The blank between { and num_1 (between } and num_n) in { $num_1 \cdots num_n$ } is imperative.
- among necessary conditions, `:value` is the reserved necessary condition which represents a value of a concept when this concept has a value. Then, the filler of *:value* is either an interval, a number, or a set of numbers.
- Infinity in the intervals of the form [x ∞] or [∞ x] is represented by *INF*: that is, [x *INF*] or [*INF* x]

A.2 Semantics of KOLA primitives

M is found in `Section` 5.3.

$$(M \ (\text{Specializes } C_1, \cdots, C_n)) = \lambda_x.(M \ \ C_1)x \wedge \cdots \wedge (M \ \ C_n)x$$
$$(M \ (\text{Vrconcept } AR \ C)) = \lambda_x.[\forall y, (M \ \ AR)xy \rightarrow (M \ \ C)y]$$
$$(M \ (\text{Min } AR \ n)) = \lambda_x.[\exists ny.(M \ \ AR)xy]$$
$$(M \ (\text{Max } AR \ n)) = \lambda_x. \sim [\exists(n+1)y.(M \ \ AR)xy]$$
$$(M \ (\text{Differentiate } ARC_1 \ ARC_2)) = \lambda_x.[\forall y, (M \ \ ARC_1)xy \rightarrow (M \ \ ARC_2)xy]$$

Consider the following about a primitive and a defined concept:

$$\begin{aligned}
&(1) \text{ For all } x, \ Concept(x) \Rightarrow (Con_1 \wedge \cdots \wedge Con_l \wedge R_1 \wedge R_2 \wedge \cdots \wedge R_m \\
&\qquad \wedge A_1 \wedge A_2 \wedge \cdots \wedge A_n \wedge RC_1 \wedge \cdots \wedge RC_p \\
&\qquad \wedge AC_1 \wedge \cdots \wedge AC_q)(x), \quad \text{where } l, m, n, p, q \geq 0 \\
&\qquad \cdots \ \textbf{(Primitive)} \\
&(2) \text{ For all } x, \ (Con_1 \wedge \cdots \wedge Con_s \wedge R_1 \wedge \cdots \wedge R_t \wedge RC_1 \wedge \cdots \wedge RC_u)(x) \\
&\qquad \Leftrightarrow Concept(x), \quad \text{where } s \geq 0 \wedge t, u > 0 \\
&\qquad \cdots \ \textbf{(Defined)}
\end{aligned}$$

Con_i denotes all the inherited necessary conditions of superconcepts of *Concept* in which the restrictions(value, number, and/or attribute to role restriction) are applied appropriately. R_j and A_k are a role and an attribute each of which is newly defined in *Concept*. RC_a and AC_b denote a role constraint and an assertional constraint which the concept *Concept* has but none of its superconcepts has. Even though we can specify nondefinitional necessary conditions in defining a defined concept, they are not sufficient. Thus, nondefinitional necessary conditions are not specified in the necessary and sufficient conditions of a defined concept. Based on the restrictions or constraints used to define a concept, the classifier computes the subsumption relationship between concepts and build the concept taxonomy without waiting for all instances of a concept to be created. For a primitive concept, the expression (1) means that for all x, if x satisfies all of the conditions in the righthand side, it is a member of *Concept*. For a defined concept, the expression (2) means that for all x, x satisfies all of the conditions in the righthand side if and only if x is a member of *Concept*.

A.3 Algorithm for instantiator and classifier

A.3.1 Classifier

The classifier in KOLA classifies concepts by comparing the structural differences between concepts based on the subsumption relationship.

A concept is placed its most specific superconcepts and most general subconcepts. Finding such places is done by the function *FindProperSuperC*.

For a given concept, say Con, it is classified by such;

```
Let SUPS be the superconcepts specified explicitly in the description of CON.
Let IRoles and IAttri be the set of roles and attributes which are inherited from
 every concept in SUP and restrictions and constraints are reflected.
Let ToTal-R and ToTal-A be the set of roles and attributes which CON has:
 ToTal-R is the union of IRoles and the roles in the decription of CON which given
  restrictions and constraints are reflected.
 ToTal-A is the union of IAttri and the attributes in the description of CON which
  given restrictions and constraints are reflected.
Let R-Cons (A-Cons) be the union of the set of all role (assertional) constraints
 inherited from superconcepts of CON and the set of role (assertional) constraints
 in the description of CON.

Then SUPS is refined by such:

 SUPS <= (FindProperSups SUPS ToTal-R ToTal-A R-Cons A-Cons)


<FindProperSups>
 Find the most specific concepts which satisfies all restrictions and constraints given
 as follows:
  TEMP-SUP <= NIL
  For each SUP in SUPS,
   Let SUP-SUBS be the set of subconcepts of SUP.
   For each SUB in SUP-SUBS,
    TEMP-SUP <= The union of TEMP-SUP and
         (if (Satisfy-SubSumption-R?)
          then (return (FindProperSups (list SUB) ToTal-R ToTal-A R-Cons A-Cons))
          else (list SUP))
  (return (Leave-Final-Sups SUPS TEMP-SUP))
```

```
< Satisfy-SubSumption-R? >
  Let ITS-R (ITS-A) be the set of all roles (attributes) which SUB has.
  Let ITS-RC (ITS-AC) be the set of all role (assertional) constraints SUB has.
  If SUB satisfies the following:
   1. ITS-R is the subset of ToTal-R, and
   2. Each R in ITS-R satisfies the subsumption relationships given in Section 5.1.2
      in order for SUB to be a subconcept of CON, and
   3. ITS-A is the subset of the union of ToTal-R and ToTal-A, and
   4. Each A in ITS-A satisfies the subsumption relationships given in 5.1.2
      in order for SUB to be a subconcept of CON, and
   5. ITS-RC is the subset of R-Cons and any constraints in R-Cons is satisfied
      in SUB in order for SUB to be a subconcept of CON, and
   6. ITS-AC is the subset of A-Cons and any constraints in A-Cons is satisfied
      in SUB in order for SUB to be a subconcept of CON
  then (return T)
  else (return NIL)


< Leave-Final-Sups >
  WORKING-SET <= NIL
  For each SUP in TEMP-SUP
    If SUP is a subconcept of a concept in SUP, or SUP is non-primitive,
      then WORKING-SET <= WORKING-SET which SUP is added into
  For each NEW-SUP in WORKING-SET
    Let INDIRTRANS be a set of necessary conditions with indirect transitivity
     of NEW-SUP
    FINAL-SUPS <= NIL
    For each INDIR in INDIRTRANS
      Let RANGE-1 be the range of INDIR of NEW-SUP
      Let THROUGH be a necessary condition related to INDIR
      SUP? <= (Sub-By-Part-Of NEW-SUP WORKING-SET-without-NEW-SUP
                      INDIR THROUGH RANGE-1)
      FINAL-SUPS <= If SUP?
                    then FINAL-SUPS which SUP? is added into
                       and Exit
                    else FINAL-SUPS
  (return FINAL-SUPS)


< Sub-By-Part-Of >
  For each ANOTHER in WORKING-SET-without-NEW-SUP
    Let INDIRT be a set of necessary conditions with indirect transitivity
     of ANOTHER
    If INDIR is in INDIRT
     then Let RANGE-2 be the range of INDIR of ANOTHER
        If the range of THROUGH of RANGE-1 is RANGE-2
         then establish the subsumption link from NEW-SUP to ANOTHER
           and (return NEW-SUP)
```

Given the instance, INS, the instantiator works by such:
Let **Templates** be the set of concepts each of which is given in the description of INS.
INS must be the member of concepts in **Templates**.
Each concept in **Templates** is the representative, not dummy. (cf. Synonyms of concepts)

Let **Prop-Val** be the set of pairs each of which consists of property name and its fillers.
Each filler is the representative, not dummy.
Each pair in **Prop-Val** can be found
either directly in the description of INS
or indirectly through symmtricity of a property or inverse property
the pair which can be found indirectly is included in **Prop-Val** vividly.

Templates is refined by such and the instantiator connects INS with each of concepts in the refined **Templates**.

```
Templates <= (FindMostSpecificCon Templates Prop-Val)
```

```
< FindMostSpecificCon >
 Let Props be the set of property names in Prop-Val.
 Current-Mom <= If Templates is nonempty,
            then Templates,
            else (FindPotentialMom Props)
 Refined-Moms <= NIL
 For each Mom in Current-Mom
   Let Mom-RAs be the set of all roles and attributes Mom has.
   Refined-Moms <= The union of Refined-Mom and
                    (ClosestMom Prop-Val Props Mom Mom-RAs)
 (return (CanbeMom? Refined-Moms))
```

```
< FindPotentialMom >
 Find and return the concepts each of which satisfies the following:
   Let Con be the concept of concern.
   Let RAs be the set of all roles and attributes Con has.
   Then, RAs is the subset of Props.
```

```
< ClosestMom >
 Let CommonProp be the intersection of Mom-RAs and Props.
 If (SatisfiableRes&Con? Mom Prop-Val)
  then, Let Subs be the set of all subconcepts of Mom
      More-Specific <= NIL
      For Each-Sub in Subs,
        Let Sub-RAs be the set of all roles and attributes Each-Sub has.
        More-Specific <= the union of More-Specific and
                 (ClosestMom Prop-Val Props Each-Sub Sub-RAs)
    If More-Specific is empty,
```

```
        then (return (list Mom))
        else (return More-Specific)
    else, Let Sups be the set of all superconcepts of Mom.
      (return (FindInSups Sups Prop-Val Props))


< FindInSups >
  More-General <= NIL
  For Each-Sup in Sups
     Let Sup-RAs be the set of all roles and attributes Each-Sup has.
     More-General <= the union of More-General and
              (ClosestMom Prop-Val Props Each-Sup Sup-RAs)
  If More-General is nonempty,
    then (return More-General)
    else, Let All-Sups be the set of superconcepts of concepts in Sups.
      (return (FindInSups All-Sups Prop-Val Props))


< SatisfiableRes&Con? >
  If  Mom-RAs is the subset of Props and
     for all properties which are found commonly both in Mom-RAs and Props,
     no fillers in Prop-Val violate any restrictions or constraints as explained
     in Section 4.
    then (return T)
    else (return NIL)


< CanbeMom? >
  Final-Moms <= NIL
  For Each-Con in Refined-Moms
     If (SubORNonPrim? EachCon Templates),
       then Final-Moms <= the union of Final-Moms and
                            (list Each-Con)
  (return Final-Moms)


< SubORNonPrim? >
  If Each-Con is the subconcept of at least one of concepts in Templates
    or a non-primitive concept,
  then (return T)
  else (return NIL)
```

A.4 Running Example

A.4.1 Terminological Knowledge

```
(defconcept strings p)
(defconcept thing p)
(defconcept numbers p)
(defconcept genders p)
(defconcept female (s genders))
(defconcept male (s genders))
(defrole :value (domain thing) (range numbers))

(defconcept person p
            (attribute name (vrc strings) (min 0))
     (attribute gender (vrc genders) (num 1))
     (attribute age (vrc numbers) (num 1))
            (attribute anatomy (vrc anat-entity) (number 1))
     (attribute occupation (vrc job) (min 0))
     (role children (vrc person) (min 0))
     (role father (vrc married-male-person) (num 1))
     (role mother (vrc married-female-person) (num 1))
     (assert (!= (father spouse name) (mother name))))

(defconcept male-person (s person)
            (role gender (vrc male)))
(defconcept female-person (s person)
            (role gender (vrc female)))
(defconcept married-person (s person)
            (role spouse (vrc married-person) (num 1)))
(defconcept married-male-person (s married-person male-person)
            (role spouse (vrc married-female-person)))
(defconcept married-female-person (s married-person female-person)
            (role spouse (vrc married-male-person)))

(defconcept doctors (s person)
            (role occupation (vrc medical-job) (min 1)))
(defconcept job p)
(defconcept medical-job p (s job))
```

```
(defconcept leukemia p (s disease))
(defconcept patient p (s person)
     (role patient-disease (vrc disease) (min 0))
     (role state-description (vrc patient-state-description) (min 0))

(defconcept patient-state-description p)
 ; it needs to be refined by, for example, including its necessary
 ; conditions.

(defconcept entity p)
(defconcept anat-entity p (s entity)
           (role anatomical-connected-to (vrc anat-entity) (min 0))
           (role input (vrc anat-entity) (min 0))
           (role output (vrc anat-entity) (max 0)))

(defconcept anatomical-entity p (s entity)
            (role anatomical-part-of (vrc anat-entity) (min 0)))
(defconcept anat-entity-by-region p (s anat-entity))
(defconcept anat-entity-by-function p (s anat-entity))
(defconcept anatomical-entity-by-region p (s anat-entity)
            (role classified-by (vrc region) (number 1)))
(defconcept anatomical-entity-by-function p (s anat-entity)
            (role classified-by (vrc function) (number 1)))

(defconcept anatomical-conduct p (s anat-entity)
           (role input (vrc anat-entity) (min 0))
           (role output (vrc anat-entity) (min 0)))

(defconcept region p)
(defconcept outer-kidney (s region))
(defconcept inner-kidney (s region))
(defconcept function p)
(defconcept anat-system p)

(defconcept urinary-system p (s anat-system))
(defconcept kidney p (s anat-entity)
            (role anatomical-part-of (vrc urinary-system))
            (role anatomical-connected-to (vrc ureter)))
(defconcept left-kidney (s kidney))
(defconcept right-kidney (s kidney))
```

```
(defconcept ureter p (s anat-entity)
            (role anatomical-part-of (vrc urinary-system))
            (role anatomical-connected-to (vrc urinary-bladder)))

(defconcept left-ureter (s ureter)
           (role input (vrc left-kidney)))
(defconcept right-ureter (s ureter)
           (role input (vrc right-kidney)))

(defconcept urinary-bladder p (s anat-entity)
            (role anatomical-part-of (vrc urinary-system))
            (role anatomical-connected-to (vrc urethra))
            (role input (vrc ureter)))
(defconcept urethra p (s anat-entity)
            (role input (vrc urinary-bladder))
            (role anatomical-part-of (vrc urinary-system)))

(defconcept cortex p (s anat-entity-by-region)
            (role classified-by (vrc outer-kidney))
            (role anatomical-part-of (vrc kidney))
            (role anatomical-connected-to (vrc medula)))
(defconcept medula p (s anat-entity)
            (role classified-by (vrc inner-kidney) (num 1))
            (role anatomical-part-of (vrc kidney)))

(defconcept nephron p (s anat-entity-by-function)
            (Role anatomical-part-of (vrc kidney)))
(defconcept collecting-tubule p (s anat-entity-by-function)
            (role anatomical-part-of (vrc kidney))
            (role anatomical-connected-to (vrc distal-tubule))
            (role input (vrc tubule)))

(defconcept glomerule p (s anat-entity-by-function)
           (role anatomical-part-of (vrc nephron))
           (role output (vrc tubule)))

(defconcept tubule p (s anat-entity-by-function anatomical-conduct)
            (role anatomical-part-of (vrc nephron))
            (role input (vrc glomerule))
```

```
            (role output (vrc collecting-tubule)))

(defconcept bowman-capsule p (s anat-entity)
            (role classified-by (vrc region) (number 1))
            (role anatomical-part-of (vrc tubule)))

(defconcept proximal-convoluted-tubule p
            (s anatomical-conduct anat-entity)
            (role classified-by (vrc region) (number 1))
            (role anatomical-connected-to (vrc bowman-capsule))
            (role anatomical-part-of (vrc tubule))
            (role input (vrc glomerule))
            (role output (vrc loop-of-henle))

(defconcept loop-of-henle p (s anatomical-conduct anat-entity)
            (role classified-by (vrc region) (number 1))
            (role anatomical-connected-to
            (vrc proximal-convoluted-tubule))
            (role anatomical-part-of (vrc tubule))
            (role input (vrc proximal-convoluted-tubule))
            (role output (vrc distal-tubule)))

(defconcept distal-tubule p (s anat-entity)
            (role classified-by (vrc region) (number 1))
      (role anatomical-part-of (vrc tubule))
      (role anatomical-connected-to (vrc loop-of-henle))
      (role input (vrc loop-of-henle))
      (role output (vrc collecting-tubule)))

(defconcept disease p)
(defconcept kidney-disease (s disease)
            (role anatomical-site (vrc kidney)))
(defconcept nephrotic-disease (s disease)
            (role anatomical-site (vrc nephron)))

(defconcept blood-vessel p (s anatomical-conduct)
           (role input (vrc blood-vessel) (min 0))
           (role output (vrc blood-vessel) (min 0)))
(defconcept artery p (s blood-vessel))
(defconcept vein p (s blood-vessel))
```

```
(defconcept systemic-artery p (s artery))
(defconcept systemic-vein p (s vein))

(defconcept renal-blood-vessel (s blood-vessel))
(defconcept renal-arterie (s artery renal-blood-vessel)
           (role input (vrc systemic-artery))
           (role output (vrc renal-vein)))
(defconcept renal-vein (s vein renal-blood-vessel)
           (role input (vrc renal-arterie))
           (role output (vrc systemic-vein)))

(defconcept body-function p)
(defconcept homeostatic-mechanism p)
(defconcept regulation-of-body-function p)
(defconcept regulation-of-blood-gases p)
(defconcept regulation-of-blood-pressure p)
(defconcept regulation-of-fluid-volume p)
(defconcept regulation-of-body-temperature p)
(defconcept regulation-of-fluid-osmolarity p)

(defconcept measurable-thing (s thing))
(defrole measure-of (domain physiologiCal-parameter)
           (range measurable-thing))
(defconcept measurable-thing p)
(defconcept Na (s measurable-thing))
(defconcept K (s measurable-thing))
(defconcept HCO3 (s measurable-thing))
(defconcept Cl (s measurable-thing))
(defconcept Ca (s measurable-thing))
(defconcept creatinine (s measurable-thing))
(defconcept PaCO2 (s measurable-thing))
(defconcept PO2 (s measurable-thing))
(defconcept Ph (s measurable-thing))
(defconcept anion-gaps (s measurable-thing))
(defconcept glucose (s measurable-thing))
(defconcept osmolarity (s measurable-thing))
(defconcept NH3 (s measurable-thing))

(defconcept patient-property p)
(defconcept heart-contraction p (s thing))
```

```
(defconcept heart-expansion p (s thing))
(defconcept oral-part (s aNat-entity))
(defconcept rectal-part (s aNat-entity))
(defconcept Physiological-parameter (s thing)
          (role :value (vrc numbers))
          (role measured-from (vrc aNat-entity))
          (role measured-sex (vrc genders)))
(defconcept blood-pressure (s PhysiologiCal-parameter)
          (attribute functionally-associated (vrc thing)))
(defconcept Systolic (s blood-pressure)
          (role functionally-associated (vrc heart-contraction))
          (role :value (vrc [ 90 140 ]))) ;  mm Hg (Hg = mercury)
(defconcept Diastolic (s blood-pressure)
          (role functionally-associated (vrc heart-expansion))
          (role :value (vrc [ 60 89 ])))
(defconcept body-temperature (s Physiological-parameter))
(defconcept oral-temperature (s body-temperature)
          (role measured-from (vrc oral-part))
          (role :value (vrc 37))) ; C 98.6F
(defconcept rectal-temperature (s body-temperature)
          (role measured-from (vrc rectal-part))
          (role :value (vrc 37.4))) ; C, 99.3F
(defconcept total-body-Na-store (s Physiological-parameter)
          (role measure-of (vrc Na))
          (role :value (vrc numbers)))
(defconcept total-body-Na (s Physiological-parameter)
          (role measure-of (vrc Na))
          (role :value (vrc numbers)))
(defconcept total-body-K (s Physiological-parameter)
          (role measure-of (vrc K))
          (role :value (vrc numbers)))
(defconcept total-body-HCO3 (s Physiological-parameter)
          (Role measure-of (vrc HCO3))
          (role :value (vrc numbers)))

(defconcept icf p)
(defconcept intracellular-fluid p)

(defconcept icf-parameter p (s Physiological-parameter)
          (role parameter-of (vrc icf)))
```

```
(defconcept icf-Ph p (s icf-parameter)
          (role measure-of (vrc Ph))
          (role :value (vrc numbers)))
(defconcept total-icf-Na p (s icf-parameter)
          (role measure-of (vrc Na))
          (role :value (vrc numbers)))
(defconcept total-icf-K p (s icf-parameter)
          (role measure-of (vrc K))
          (role :value (vrc numbers)))
(defconcept total-icf-HCO3 p (s icf-parameter)
          (role measure-of (vrc HCO3))
          (role :value (vrc numbers)))
(defconcept total-icf-Cl p (s icf-parameter)
          (role measure-of (vrc Cl))
          (role :value (vrc numbers)))
(defconcept total-icf-Ca p (s icf-parameter)
          (role measure-of (vrc Ca))
          (role :value (vrc numbers)))

(defconcept renal-parameter p (s Physiological-parameter))
(defconcept gfr p (s renal-parameter))
(defconcept renal-function p (s renal-parameter))

(defconcept ecf p)
(defconcept ecf-parameters p (s Physiological-parameter)
          (role parameter-of (vrc ecf)))
(defconcept total-ecf-volume p (s ecf-parameters)
          (role :value (vrc numbers)))
(defconcept total-ecf-Na p (s ecf-parameters)
          (role measure-of (vrc Na))
          (role :value (vrc numbers)))
(defconcept total-ecf-K p (s ecf-parameters)
          (role measure-of (vrc K))
          (role :value (vrc numbers)))
(defconcept total-ecf-HCO3 p (s ecf-parameters)
          (role measure-of (vrc HCO3))
          (role :value (vrc numbers)))
(defconcept total-ecf-Cl p (s ecf-parameters)
          (role measure-of (vrc Cl))
          (role :value (vrc numbers)))
```

```
(defconcept total-ecf-Ca p (s ecf-parameters)
          (role measure-of (vrc Ca))
          (role :value (vrc numbers)))

(defconcept serum p)
(defconcept blood p)
(defconcept blood-parameters p (s Physiological-parameter)
          (role parameter-of (vrc blood)))
(defconcept serum-parameter p (s renal-parameter blood-parameters)
          (role parameter-of (vrc serum)))
(defconcept wbc p (s blood-parameters))
(defconcept rbc p (s blood-parameters))

(defconcept serum-Na-concentration p (s serum-parameter)
          (role measure-of (vrc Na))
          (role :value (vrc [ 136 146 ]) (data (unit meq/l))))
(defconcept serum-K-concentration p (s serum-parameter)
          (role measure-of (vrc K))
          (role :value (vrc [ 3.5 5.1 ]) (data (unit meq/l))))
(defconcept serum-HCO3-concentration p (s serum-parameter)
          (role measure-of (vrc HCO3))
          (role :value (vrc [ 21 29 ]) (data (unit meq/l))))
(defconcept arterial-serum-HCO3-concentration p
          (s serum-HCO3-concentration)
          (role measure-of (vrc HCO3))
          (role :value (vrc [ 21 28 ]) (data (unit meq/l)))
          (role measured-from (vrc artery)))
(defconcept venious-serum-HCO3-concentration p
          (s serum-HCO3-concentration)
          (role measure-of (vrc HCO3))
          (role :value (vrc [ 22 29 ]) (data (unit meq/l)))
          (role measured-from (vrc vein)))
(defconcept serum-Cl-concentration p (s serum-parameter)
          (role measure-of (vrc Cl))
          (role :value (vrc [ 98 106 ]) (data (unit meq/l))))
(defconcept serum-Ca-concentration p (s serum-parameter)
          (role measure-of (vrc Ca))
          (role :value (vrc [ 2.1 2.55 ]) (data (unit mmol/l))))
(defconcept serum-creatinine-concentration p (s serum-parameter)
          (role measure-of (vrc creatinine))
```

```
          (role :value (vrc [ 0.17 0.93 ]) (data (unit meq/l))))
(defconcept male-serum-creatinine-concentration p
           (s serum-creatinine-concentration)
           (role measure-of (vrc creatinine))
           (role :value (vrc [ 0.17 0.70 ]) (data (unit meq/l)))
           (role measured-sex (vrc male)))
(defconcept female-serum-creatinine-concentration p
            (s serum-creatinine-concentration)
           (role measure-of (vrc creatinine))
           (role :value (vrc [ 0.35 0.93 ]) (data (unit meq/l)))
           (role measured-sex (vrc female)))
(defconcept serum-PaCO2 p (s serum-parameter)
          (role measure-of (vrc PaCO2))
          (role :value (vrc [ 32 48 ]) (data (unit meq/l))))
(defconcept male-serum-PaCO2 p (s serum-PaCO2)
           (role measure-of (vrc PaCO2))
           (role :value (vrc [ 35 48 ]) (data (unit meq/l)))
           (role measured-sex (vrc male)))
(defconcept female-serum-PaCO2 p (s serum-PaCO2)
           (role measure-of (vrc PaCO2))
           (role :value (vrc [ 32 45 ]) (data (unit meq/l)))
           (role measured-sex (vrc female)))
(defconcept serum-PO2 p (s serum-parameter)
          (role measure-of (vrc PO2))
          (role measured-from (vrc artery))
          (role :value (vrc [ 83 108 ] (data (unit mm Hg)))))
(defconcept serum-Ph p (s serum-parameter)
          (role measure-of (vrc Ph))
          (role :value (vrc [ 7.35 7.45 ]) (data (unit meq/l))))
(defconcept anion-gap p (s serum-parameter)
          (role measure-of (vrc anion-gaps))
          (role :value (vrc [ 7 16 ]) (data (unit mmol/L))))
(defconcept serum-glucose-concentration p (s serum-parameter)
          (role measure-of (vrc glucose))
          (role :value (vrc [ 70 105 ]) (data (unit mg/dl))))
(defconcept serum-osmolarity p (s serum-parameter)
          (role measure-of (vrc osmolarity))
          (role :value (vrc [ 275 295 ]) (data (unit mOsmol/Kg))))

(defconcept urinary-parameter p (s Physiological-parameter))
```

```
(defconcept urine-output p (s urinary-parameter))
(defconcept urine-osmolarity p (s urinary-parameter))
(defconcept urinary-Ph p (s urinary-parameter))
(defconcept urinary-NH3-concentration p (s urinary-parameter))
(defconcept urinary-electrolyte p (s urinary-parameter))
(defconcept urinary-Na p (s urinary-electrolyte)
          (role measure-of (vrc Na))
          (role :value (vrc [ 40 220 ]) (data (unit mmol/L))))
(defconcept urinary-K p (s urinary-electrolyte)
          (role measure-of (vrc K))
          (role :value (vrc [ 25 125 ]) (data (unit mmol/L))))
(defconcept urinary-Cl p (s urinary-electrolyte)
          (role measure-of (vrc Cl))
          (role :value (vrc [ 110 250 ]) (data (unit mmol/L))))
(defconcept urinary-HCO3 p (s urinary-electrolyte)
          (role measure-of (vrc HCO3))
          (role :value (vrc 0) (data (unit mmol/L))))

(disjoint genders :name genders (female male))
(disjoint region :name region (outer-Kidney inner-Kidney))
(Transitive anatomical-part-of)
(Transitive anatomical-connected-to :direction bidirectioNal)
(indirTrans anatomical-site :via anatomical-part-of)
(symmetric spouse)
(synonyms-c icf intracellular-fluid)
(synonyms-c anatomical-entity anat-entity)
(synonyms-c anatomical-entity-by-function anat-entity-by-function)
(synonyms-c anatomical-entity-by-region  anat-entity-by-region)
(prop-assert children ::= spouse children)
(prop-assert sibling ::= father children)
(prop-assert sibling ::= mother children)
(prop-assert mother ::= father spouse)
(prop-assert father ::= mother spouse)
```

A.4.2 Assertional Knowledge

```
(defins male (:Instanceof male))
(defins female (:Instanceof female))
(defins surgeon (:Instanceof Medical-job))
(definstance Jason (:Instanceof person)
  (name "Jason Lee")
  (children (:all 2) Kib Brian)
  (gender male)
  (spouse Mary)
  (occupation :canbemore surgeon)
  (age 40))
(defins Mary (:Instanceof person)
  (name "Mary Lee")
  (gender female)
  (age 35))
(defins Kib (:Instanceof person)
  (gender male)
  (name "Kib Lee")
  (father Jason))
(defins Brian-leuKemia (:Instanceof leuKemia))
(defins Brian (:Instanceof person)
  (gender male)
  (name "Brian Lee")
  (mother Mary)
  (patient-disease Brian-leuKemia))

(DetailFR Children (In Jason) First-son (Fillers Kib))
(DetailFR Children (In Jason) Second-son (Fillers Brian))
```

A.5 KOLA operators

1. Symmetry
(Symmetry *ra-name*) *Macro*

It lets the system know that a necessary condition *ra-name* is symmetric.
It tells I-World that if a concept with *ra-name* is instantiated, say i_1, and $i_1 RA i_2$, then $i_2 RA i_1$ is also true for an instance i_2.

Thus, although it is known only the fact that the instance i_1 has i_2 as *ra-name*'s filler, I-World can infer and represent vividly the fact that i_2 has i_1 as *ra-name*'s filler. Such vivid representation of knowledge affects the connection of an instance to its most specific concepts each of which it should belong to.

2. Transitive
(Transitive *ra-name* [`:direction` bi(direction)]) *Macro*

It lets the system know that a necessary condition *ra-name* has the transitive property. `:direction bi` is to tell the system that *ra-name* is both transitive and symmetric. This information is to be used to lead Question-Answerer to the right direction to get to the answer efficiently. (*cf*, see `Section` 5.1.3)

3. indirect transitive
(Ind[irect]Trans[itive] *ra-name* `:via` *via*) *Macro*

The system can be told that *ra-name* has the indirect transitive property. When Question-Answerer is solving a problem related to *ra-name*, this information lets Question-Answerer know that it may be necessary to search thorough *via* as well as *ra-name*. (*cf*, see `Section` 5.1.3)

4. Inverse of a necessary condition
(InverseAR RA_1 RA_2) *Macro*

The system can be told that there is the inverse relation between RA_1 and

RA_2. When an instance with RA_1 or RA_2 is created, the instantiator represents vividly knowledge which is implicit but can be inferred by using this information. LiKe *Symmetry*, it also has the influence on determining the connection of an instance to its most specific concepts. Suppose a concept with a necessary condition *ra-name* is instantiated, say i_1 and $i_1 RA_1 i_2$. Then, $i_2 RA_2 i_1$ is true.

5. The way to get there

(**prop-assert** *interested* := *chain of necessary conditions*) *Macro*

It helps Question-Answerer to know that what we are interested can be reached through other ways indirectly: the fillers of *interested* in an instance can be reached though a given chain. *interested* may or may not be a property defined in an instance explicitly. Reconsider the example in `Section` 5.1.4. Consider

`Children := spouse Children.`

For an instance *Ins*, the fillers of *Children* in *Ins* can be reached by such:

1. Find the fillers of *spouse* of *Ins*.
2. for each filler found in (1), find the fillers of *Children*.

On the other hand, consider

`(Sibling := father children)` or `(Sibling := mother children).`

Although `Sibling` is not defined as a necessary condition for any concept in C-World, it can be reached through the `chain of necessary conditions`. (See `Section` 5.1.4.)

(**undo-prop-assert** *interested* ::= *chain to be deleted*) *Macro*

Undo the defined way to get a solution. For example, we can make the effect of `(prop-assert Children := spouse Children)` undone, using `(undo-prop-assert Children ::= spouse Children).`

6. Detailed Filler References

(1) (**DetailFR** *Prop* (**In** *Ins*) *Ref* (**Fillers** *the list of Fillers*)) *Macro*

(2) (**DetailFR** *Prop1* (**In** *Ins*) **:i-Sort-by** *Prop2*) *Macro*
(3) (**DetailFR** *Prop1* (**In** *Ins*) **:d-Sort-by** *Prop2*) *Macro*

(1) lets KOLA know that *the list of Fillers* among fillers of the property *Prop* in the instance *Ins* can be reached through the detailed references *Ref.* (2) and (3) are the constructs to tell KOLA that when fillers of the property *Prop1* in the instance *Ins* are specified, they need to be Sorted according to the value of *Prop2* each of which has in increasing (2) or decreasing (3) order. For example, consider the following:

(**DetailFR** *Children* (**In** *Jason*) **:d-Sort-by** *age*)

It informs that when Jason's children are specified, they are sorted according to their age in decreasing order.

7. disjointness Class
(**disjoint** [*disjointed*] **:name** *name* ($Con_1 \cdots Con_n$)), where $n \geq 2$ *Macro*

It tells the system about a set of concepts for which there are no common instances. If *disjointed* is specified, it is the disjointed concept. The set of $Con_1 \cdots Con_n$ is the corresponding disjointness class. Each disjointness classe has to be associated with a unique name, to facilitate several useful operations on disjointness classes, including redefining a disjointness class. Such a name follows the Keyword `:name`. See `Section` 5.1.4 for details.

(**redefine-disjoint** [*c-name*] **:name** *name* ($Con_1 \cdots Con_n$)), where $n \geq 0$ *Macro*

This function is for redefining or deleting a disjointness class which was already defined. If $n = 0$, the disjointness class with the name *name* is deleted. If $n \geq 1$, the disjointness class with the name *name* is replaced by given $Con_1 \cdots Con_n$.

(**FindnamesOfdisjointnessClass** [*disjointedC*]) *Macro*

This function shows names for all disjointness classes defined. If *disjointedC* is specified, names of disjointness classes defined for this concept are returned. Otherwise, names of disjointness classes defined without being associated with any disjointed concept are returned.

(ShowdisjointnessClass [**:disjointed** *c-name*] [**:name** *name*]) *Macro*

This function returns the corresponding disjointness class(es). When both *c-name* which follows the keyword `:disjointed` and *name* which follows `:name` are specified, the disjointness class with the name *name* of the disjointed concept *c-name* are returned. When only *c-name* which follows `:disjointed` is given, all disjointness classes for this concept are returned. When only *name* which follows `:name` is specified, a disjointness class with the name *name* is returned: in this case, it does not have any corresponding disjointed concept. When no options are specified, all disjointness classes each of which does not have any corresponding disjointed concepts are returned.

8. cover class

(cover *c-name* **:name** *name* $(Con_1 \cdots Con_n)$), where $n \geq 2$ *Macro*

It tells the system about a set of concepts which exhaust its covered set. A set of coverings also has to be associated with a name: its name is specified by following `:name`.

(redefine-cover *c-name* **:name** *name* $(Con_1 \cdots Con_m)$), where $m \geq 0$ *Macro*

This function is for redefining or deleting a set of coverings which was already defined. If $m = 0$, the set of coverings with the name *name* is deleted. If $m \geq 1$, the set of coverings with the name *name* is replaced by given $Con_1 \cdots Con_m$.

(FindnamesOfcoverings *coveredC*) *Macro*

This function returns names for sets of coverings of the covered concept *coveredC*.

(Showcoverings :covered *c-name* [**:name** *name*]) *Macro*

This function returns the corresponding set of coverings of the covered concept *c-name*. When *name* which follows `:name` is specified, the set of coverings with the name *name* of the covered concept *c-name* is returned. When *name* is not given, all sets of coverings of the covered concept *c-name* are returned.

9. Synonyms

(1) (**Synonyms-C** $Con_1 \cdots Con_n$) where $n \geq 2$ *Macro*
(2) (**Synonyms-Ins** $Ins_1 \cdots Ins_n$) $n \geq 2$ *Macro*

(1) lets the system know about synonyms of concepts, while (2) about synonyms of instances. Details can be found in `Section` 5.1.4.

A.6 KOLA's ask operators

(**Is** *Con-Ins-2 RA-name* **of** *Con-Ins-1*) *Macro*

This is for asking a query such as *"Is the filler of RA-name in Con-Ins-1 Con-Ins-2?"*

Con-Ins-1 or *Con-Ins-2* can be either a concept or an instance. If the system decides that *Con-Ins-1* is a concept, this query is asking for a range of its necessary condition, *RA-name*. If the system determines that *Con-Ins-1* is an instance, then this query is asking for a filler of its property, *RA-name*. In the case where *Con-Ins-1* is an instance, *RA-name* can be of the form *(RA-name Detailed-Filler-Reference)* to ask about the filler which can be reached through this detailed filler reference. To reach as correct an answer as possible, KOLA has the ability to use information about symmetry, inverse relation, transitivity, and so on. For details, see `Section` 5.4.

(**WhatIs :C[:Ins]** *RA-name* **of :C[:Ins]** *Domain*) *Macro*

This is for asking a range (fillers) of a necessary condition (property) in a concept (an instance) based on the flag whose value is either **:C** or **:Ins**. If it is asking for fillers of an instance's property, *RA-name* can be of the form either *(RA-name detailed-filler-reference)* or *(RA-name* **:sort-by** *indiCator* **:n-st** *num*). In the second case, each of fillers of an instance's *RA-name* are sorted by the value of its property *indicator*. Among them, the *num*-st filler of *RA-name* will be retrieved.

(**WhereIBelongTo** ($prop_1 \cdots prop_n$) ($con_1 \ldots con_m$)) *Macro*

Return the most appropriate concepts each of which a given instance should belong to.

($prop_1 \cdots prop_n$) is the list of $prop_i$'s each of which is of the form either P_{name} or (P_{name} *fillers*). It consists of information on properties that an instance of concern has. ($con_1 \ldots con_m$) is the list of concepts which we are sure an instance of concern belongs to. For an instance to be a member of a primitive concept, such a fact has to be specified explicitly. For example, if an instance belongs to the primitive concept *Person*, *Person* has to appear in the list of ($con_1 \ldots con_m$).

(**FindInstanceWith** ($Prop_1 val_{11} \cdots val_{1m}$ [:all]) $\cdots$ ($prop_n val_{n1} \cdots val_{nm}$ [:all]))
Macro

Return the instance which satisfies the given specification. Find the instance with the properties $Prop_i, i = 1 \cdots n$ which has fillers $val_{i1} \cdots val_{ij}$, $j = 1 \cdots m$. **:all** is the keyword to tell the system that fillers given in the specification are all fillers which an instance can have as fillers of a property of concern.

A.7 KOLA functions

(**ClassifiCation** [*option*]) *Function*

Return the concept taxonomy after Classifying concepts which were defined but unclassified. If *option* is not specified, *Never-mind* strategy for checKing the violation of disjointness Classes is selected. *option* can be either `t` or `x` where `t` is the boolean constant to represent `true` and `x` is a Natural number to control the frequency of checKing the violation of disjointness Classes. If *option* is `t`, coherence of disjointness Classes is checKed whenever 50 concepts – by default – are Classified.

(**instantiation**) *Function*

Establish instantiation linKs between an instance and its most appropriate con-

cept(s) and builds the instance networK.

(reinit) *Function*

Reinitialize the system.

(:c *C-name***)** *Macro*

Return the structure of the concept whose name is *C-name*.

(:ins *I-name***)** *Macro*

Return the structure of the instance whose name is *I-name*.

(PPConcept *Con***)** *Function*

Return information about the concept *Con*. After definitioNal information about *Con* is returned, the rest of information about it can be returned on demand.

In the dynamic lisp listener in Symbolics, **Show Concept** shows the same information as `PPConcept` does.

(PPInstance *Ins***)** *Function*

Return information about the instance *Ins*, including its most appropriate concepts.

In the dynamic lisp listener in Symbolics, **Show Instance** shows the same information as `PPInstance` does.

(PPRole *role***)** *Function*

Return information about the role *role*.

In the dynamic lisp listener in Symbolics, **Show Role** shows the same information as `PPRole` does.

(**PPRoleDomain** *role domain*) *Function*

Return information about the role *role* in the concept *domain*.

In the dynamic lisp listener in Symbolics, **Show Role** shows the same information as `PPRoleDomain` does.

(**PPAttribute** *attribute*) *Function*

Return information about the attribute *attribute*.

In the dynamic lisp listener in Symbolics, **Show Attribute** shows the same information as `PPAttribute` does.

(**PPAttributeDomain** *attribute domain*) *Function*

Return information about the attribute *attribute* in the concept *domain*.

In the dynamic lisp listener in Symbolics, **Show Attribute** shows the same information as `PPAttributeDomain` does.

(**PPIData-C** *Con*) *Function*

Returns a concept *Con*'s inherited data with pretty printing format.

(**IData-C** *Con*) *Function*

Returns a concept *Con*'s inherited data without pretty printing format.

(**PPData-C** *Con*) *Function*

Returns a concept *Con*'s noninherited data with pretty printing format.

(**data-C** *Con*) *Function*

Returns a concept *Con*'s noninherited data without pretty printing format.

(**PPIData-R** *Role*) *Function*

Returns a role *Role*'s inherited data with pretty printing format.

(**IData-R** *Role*) *Function*

Returns a role *Role*'s inherited data without pretty printing format.

(**PPData-R** *Role*) *Function*

Returns a role *Role*'s noninherited data with pretty printing format.

(**data-R** *Role*) *Function*

Returns a role *Role*'s noninherited data without pretty printing format.

(**PPIData-A** *Attri*) *Function*

Returns an attribute *Attri*'s inherited data with pretty printing format.

(IData-A *Attri*) *Function*

Returns an *Attri*'s inherited data without pretty printing format.

(PPData-A *Attri*) *Function*

Returns an *Attri*'s noninherited data with pretty printing format.

(data-A *Attri*) *Function*

Returns an attribute *Attri*'s noninherited data without pretty printing format.

(KOLA-Taxonomy *Con*) *Function*

Draws the concept taxonomy whose root is the concept *Con*. The level of the concept taxonomy is 3 by default.

In the dynamic lisp listener in Symbolics, **Show Taxonomy** shows the same taxonomy as `KOLA-Taxonomy` does.

(Subconcept-p Con_1 Con_2) *Macro*

check to see if Con_1 is a subconcept of Con_2.

(Superconcept-p Con_1 Con_2) *Macro*

check to see if Con_1 is a superconcept of Con_2.

(TellMeSuperconceptsof *Con*) *Macro*

Return all immediate superconcepts of *Con*.

(TellMeSubconceptsof *Con*) *Macro*

Return all immediate subconcepts of *Con*.

(TellMeAllSubconceptsof *Con*) *Macro*

Return all subconcepts of *Con*. To find it, the subtree of *Con* in the concept taxonomy has to be searched. It implies its cost is exponential.

(UnDoDefconcept *Con*) *Macro*

Delete the concept *Con* which was already defined. For simple implementation's sake, consistency after this function is performed is not checked. Thus, it is recommended to undo the definition of a concept before classification.

(storeInstanceFiller *ins prop Fillers*) *Macro*

This function is useful when fillers of *ins*'s *prop* is obtained by computing other functions. For example, suppose fillers of this property is the result of a formula, say $(+\ (*\ r\ 10)\ a)$, where r and a are another values. Suppose we are using common-Lisp, then we can write the function, `Ins-fill`, to compute and put fillers into *prop*'s as follows:

```
(defun Ins-fill (ins prop r a)
    (let ((fillers (Current-Formula r a)))
      (storeInstanceFiller ins prop fillers)))
```

Where `Current-Formula` is the function to compute $(+(*r10)a)$.
$< NB >$, it is recommended that this function should be carried out before the instantiation. Although it is allowed after instantiation, the checking of the consistency disturbance due to adding new fillers is not performed.

Bibliography

[Bobrow 77] Bobrow, D & T. Winograd, "KRL: Knowledge Representation Language", *cognitive science* Vol 1, 1977.

[Brachman 77] Brachman, R.J., " A Structural Paradigm for Representing Knowledge", PhD Dissertation Harvard University, 1977.

[Brachman 79] Brachman, R.J., "A Structural Paradigm for Representing Knowledge", Norwood NJ, 1979.

[Brachman 80] Brachman, R.J., "On the epistemological status of Semantic Networks", Readings in knowledge representation, 1980.

[Brachman 83] Brachman, R.J., "KRYPTON: Integrating Terminology and Assertion", AAAI 83, 1983.

[Brachman 84] Brachman, R.J., "The Tractability of Subsumption in Frame-Based Description Language", AAAI-84, 1984.

[Brachman 85] Brachman, R.J., "An Overview of The KL-ONE: Knowledge Representaion System", *cognitive science* 9, 1985.

[Dennet 81] D. Dennet, "Intentional System", Mind Design, MIT Press, 1891.

[Fahlman 77] Fahlman, SE, "A System for Representing Real World Knowledge", PhD Dissertation MIT, 1977.

[Fox 79] Fox, M.S., "On Inheritance in Knowledge Representation", Proceedings of IJCAI pp 282 -284, 1979.

[Fox 84] Fox, M.S., "Issues in Knowledge Representation for Project Management", Workshop on Principles of Knowledge-Based Systems, 1984.

[Goldstein 76] Goldstein, I & B. Roberts, "NUDGE, A Knowledge-Based Scheduling System", Fifth IJCAI, 1976.

[Haase 87] Haase Jr., K.W., "TYPICAL An implemented Approach to Type Specification and Inference with Applications to Artificial Intelligence", MIT, 1987.

[Hayes85] Hayes, P.J., "The Second Naive Physics Manifesto", *Formal Theories of the Commonsense World*, Hillsdale, N.J., 1985.

[Hendrix 75] Hendrix, G.G., "Expanding the Utility of Semantic Networks through Partitioning", Fourth IJCAI, 1975.

[Med-Dictionary] Laurence Urdang Associates, "The Bantam Medical Dictionary", BANTAM Books, 1982.

[Lenet 76] Lenet, D, "An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search", PhD Dissertation Standford University, 1976.

[Levesque 84] Levesque, H.J., "A Fundamental Tradeoff in Knowledge Representation and Reasoning", Proc. CSCSI-84, 1984.

[Levesque 86a] H.J. Levesque, *"Making Believers out of Computers"*, Artificial Intelligence, 1986.

[Levesque 86b] Levesque, H.J., "Knowledge Representation and Reasoning", Annual Reviews Computer Science, Annual Reviews Inc., 1986.

[Martin 81] Martin, W.A., and Szolovits, P., "Semantic Networks in LISP: Fundamental Concetps and a Specific Implementation", MIT, 1981.

[Minsky 75] Minsky, M., "A Framework for Representing Knowledge" The Psychology of Computer Vision – McGraw-Hill, 1975.

[Moser 83] Moser, M.G., "An Overview of NIKL ", BBN Tech. Report 5421, 1983.

[Nebel 88] Nebel, B., "Computational Complexity of Terminological Reasoning in BACK", *Artificial Intelligence 34*, 1988.

[Newell 81] Newell, A., "The knowledge level", *Artificial Intelligence*, 1981.

[Nilsson 80] Nilsson, N.J., "Principles of Artificial Intelligence", Tioga Publishing Company, 1980.

[Patel-Schneider 84] Patel-Schneider, P.F., Brachman, R.J., and Levesque, H.J., "ARGON: Knowledge Representaion meetes Information Retrieval", Proceedings of the First CAIA, 1984.

[Patel-Schneider 84] Patel-Schneider, P.F., "Small can be Beautiful in Knowledge Representation", IEEE Workshop on Principles of Knowledge-Based Systems, 1984.

[Patil] R.S. Patil, "Causal representation of patient illness for electrolyte and acidbase diagnosis", TR-267, MIT, 1981.

[Pigman 84] Pigman, V, "The Interaction Between Assertional and Terminological Knowledge in Krypton", IEEE Workshop on Principles of Knowledge-Based Systems, 1984.

[Quillian 66] Quillian, M.R., "Semantic Theory", Tech. Report AFCRL-66-189 of BBN, 1966.

[Rich 85] Rich, C., "The layered architecture of a system for reasoning about problems", IJCAI-85, 1985.

[Schbert 76] Schubert, L.K., "Extending the Expressive Power of Semantic Networks", AI Vol 7 pp 163 - 198, 1976.

[Schomolze 83] Schmolze, J.G., & Lipkis, T.A., "Classification in the KL-ONE knowledge representation system", In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983.

[Sowa 84] Sowa, J.F., "Conceptual Structures", Addison-Wesley, 1984.

[Stefik 79] Stefik, M, "An Examination of a Frame-Structured Representation System", Sixth IJCAI, 1979.

[Stickel 82] Stickel, M.E., "A Nonclausal Connection-Graph Resolution Theorem-Proving Program", AAAI-82, 1982.

[Stickel 83] Stickel, M.E., "Theory Resolution: Building in Nonequational Theories", AAAI-83, 1983.

[Vilain 85] Vilain, M.,B., "The restricted language architecture of a hybrid representation system", Proceedings of the Ninth International Joint Conference on Artificial Intelligenc, 1985.

[Woods 75] Woods, W.A., "What's in a Link : Foundations for Semantic Networks", Academic Press, 1975.

[Woods 86] Woods, W.A, "Important Issues in Knowledge Representation", *Proc. of the IEEE Vol-74*, 1986.

[Wright 84] Wright, J.M., Fox M.S., & Adam,D., "SRL/2 Users Manual", Tech. Report of CMU, 1984.